Application of Definability to Query Answering over Knowledge Bases

by

Taras Kinash

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Answering object queries (i.e. *instance retrieval*) is a central task in *ontology based data access (OBDA)*. Performing this task involves reasoning with respect to a knowledge base $\mathcal{K}$ (i.e. ontology) over some description logic (DL) dialect $\mathcal{L}$. As the expressive power of $\mathcal{L}$ grows, so does the complexity of reasoning with respect to $\mathcal{K}$. Therefore, eliminating the need to reason with respect to a knowledge base $\mathcal{K}$ is desirable.

In this work, we propose an optimization to improve performance of answering object queries by eliminating the need to reason with respect to the knowledge base and, instead, utilizing cached query results when possible. In particular given a DL dialect $\mathcal{L}$, an object query C over some knowledge base $\mathcal{K}$ and a set of *cached query results* $S = \{S_1, \ldots, S_n\}$ obtained from evaluating past queries, we rewrite C into an equivalent query D, that can be evaluated with respect to an empty knowledge base, using cached query results $S' = \{S_{i_1}, \ldots, S_{i_m}\}$, where $S' \subseteq S$. The new query D is an *interpolant* for the original query C with respect to $\mathcal{K}$ and S. To find D, we leverage a tool for enumerating interpolants of a given sentence with respect to some theory. We describe a procedure that maps a knowledge base $\mathcal{K}$, expressed in terms of a description logic dialect of first order logic, and object query C into an equivalent theory and query that are input into the interpolant enumerating tool, and resulting interpolants into an object query D that can be evaluated over an empty knowledge base.

We show the efficacy of our approach through experimental evaluation on a *Lehigh University Benchmark (LUBM)* data set, as well as on a synthetic data set, *LUBMMOD*, that we created by augmenting an *LUBM* ontology with additional axioms.

Acknowledgements

This work would not be possible without the support of many people. I owe deepest gratitude to my supervisors, Grant Weddell and David Toman. Their wise advise, knowledge and enthusiasm helped me through every day of my masters studies.

My gratitude also goes to my readers, Professors Peter Van Beek and Richard Trefler, for their useful feedback and constructive insights, and to my collaborators, Dr. Jiewen Wu, Dr. Alexander K. Hudek and Mohamed Sabiri, for their numerous contributions to this work.

I am forever indebted to my colleagues in the database research group, as well as faculty and staff members of the Cheriton School of Computer Science, for making my academic journey over the past two years pleasant and eventful.

I would also like to thank all my friends for their support over the past two years.

Last, but, by far, not least, I would like to thank my parents, Vasyl and Luba Kinash, and my brother, Nazar Kinash, for their endless love, understanding and faith in me.

Dedication

I would like to dedicate this thesis to my parents, Vasyl and Luba Kinash. Their love, support and faith gave me strength every step of the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Knowledge representation, storage and manipulation is a basic problem in computer science. The introduction of relational data model in the 1970's [15] provided an initial solution that has worked very well for many applications. However, there has been a drastic increases in volume and sophistication of data that needs to be maintained and manipulated, for which relational technology is no longer sufficient in terms of expressive power and performance. At the heart of relational technology is *relational algebra*, which is essentially *first order logic (FOL)*. A full review of FOL is beyond the scope of this thesis, however, a brief overview of first order predicate logic, which is important for this work, can be found in Appendix A. One particular limitation of the relational data model is an assumption of complete knowledge about the application domain, which has become problematic with current data sources, for example sources underlying the *open data initiative*, like DBpedia [8]. To help address this, a new paradigm of OBDA, that accommodates incomplete knowledge, has gained popularity in recent years [17, 33, 45]. The idea of OBDA is to define an ontology in some expressive language over the actual data, which may, in

turn, be stored in a variety of formats, including in the form of a legacy relational database. Answering user queries in OBDA systems may require reasoning over the underlying knowledge base. In this thesis, the term "knowledge base" is used interchangeably with a term *ontology*. Therefore, a combination of expressive power and complexity of reasoning must be considered when selecting a language for the ontology in an OBDA system. First order logic is very powerful in terms of expressiveness, but it has rather high complexity for reasoning; in fact, checking logical consequence is semi-decidable in FOL. Therefore, more attention has been directed to decidable fragments of FOL, like *description logics (DL)*. Description logics date back to the 1970's [9] and provide a powerful framework to represent ontologies and perform reasoning tasks over those ontologies. Many different DL dialects exist at this time; they provide a spectrum of compromise between expressivity and performance of reasoning.

The past few decades have seen the emergence of the Internet. In modern times, the volume and variety of data on the web is immense. As a result, the *semantic web* [32, 58] movement appeared and grew in importance. The idea behind this movement is to standardize data on the web in order to make it easier to parse and understand for machines; to create the so called "web of data". The semantic web builds on the *resource description framework (RDF)* [28]. RDF is a data model that represents information in the form of *triples*: subject-predicate-object. Informally, we can think of a subject as a resource, an object as an attribute or property of the resource, and a predicate as a relationship between subject and object. RDF is particularly suitable for representing information on the web of data. One can ask questions about data stored in RDF through the *SPARQL* query language [47]. The key construct in SPARQL is a *basic graph pattern (BGP)*, which is comparable to "SELECT/FROM/WHERE" fragment of SQL. BGPs define conditions that must be satisfied by each element in the result set. In most cases, evaluating BGPs can

be mapped to query answering in description logics (with some exceptions, see SPARQL reference [47] for details), which makes DLs a particularly important resource in the semantic web. Description logics became particularly important in SPARQL 1.1 [4], where *entailment regimes* were introduced [3]. Essentially, entailment regimes control the expressivity of ontology languages used for reasoning. Further, building on the RDF data model, the *web ontology language* (OWL 2) was created [40, 43]. Essentially, OWL 2 adds more features and capabilities to the ontology language for web data. Description logics are a big part of OWL 2. For example, most of OWL 2 can be captured by the expressive $\mathcal{SROIQ}(\mathbf{D})$ DL dialect. Also, OWL 2 supports alternative *profiles* [1] which correspond to less expressive DL dialects that have better complexity for reasoning tasks.

Description logics are expressive decidable fragments of FOL. However, with the current volume of data that needs to be maintained, reasoning in ontology languages needs to be efficient, and not only decidable. DLs offer a variety of reasoning tasks, but most of these are rather inefficient. For example, even for the $\mathcal{ALC}$ DL dialect, instance retrieval and conjunctive query answering with respect to a knowledge base $\mathcal{K}$ is a co-NP complete task in the size of the data. Complexity is even worse when the size of the knowledge base is considered. For this reason, research to improve performance of query answering for description logics is ongoing. The two main areas of focus are to find heuristics for reasoning that work well for practical cases [35], or to create more DL dialects that have enough expressive power for many practical applications, but that are less expressive than more common DLs, and for which reasoning is more efficient. Utilizing cached query results is another example of the former approach. Most modern DL reasoners do not support caching results [61], however there has been some work related to cached query results in DLs [27]. Also, caching previous results has been extensively studied in other areas of computing, including relational databases [52, 38]; results of this work can be borrowed

and/or extended to the DL case. An example of the latter approach is the DL-Lite family of languages [14] for which conjunctive query answering can be done in PTIME, in the size of the data, and is NP-complete in the size of the knowledge base.

Slow performance of query answering tasks over DL knowledge bases $\mathcal{K}$ serves as the motivation for this work. Since we are facing rather high theoretical bounds for query answering with respect to $\mathcal{K}$, the main idea for our approach is to eliminate the need to reason with respect to $\mathcal{K}$ and to instead reason with respect to an empty knowledge base when possible. To do this, we attempt to rewrite a query into a different format for which there is no need for additional facts from $\mathcal{K}$ in order to find the set of results. Our hope is that the overhead of this rewriting with respect to $\mathcal{K}$ is small relative to the case of evaluating the original query with respect to $\mathcal{K}$.

1.1 Contributions

This work addresses a problem of improving query answering time over a DL ontology by eliminating the need to reason with respect to an initial knowledge base. This thesis is largely based on the concepts of *definability*, introduced by Willem Beth [11], *interpolation*, introduced by William Craig [16], and *interpolant enumeration*[57]. In particular, the contributions are as follows:

1. We propose a procedure that applies definability and interpolant enumeration to generate a rewriting of a user query, into an equivalent query that does not require reasoning with respect to a knowledge base $\mathcal{K}$ for its evaluation. Although the procedure is general, we demonstrate how to use it in the cases when cached query results can be utilized for query answering. Our procedure allows to compute a more specific

set of relevant cached query results that is required to answer the original query.

2. We evaluate our approach with a FOL interpolant enumeration tool and DL reasoner, and contrast the results against a brute force method for producing the rewriting. We do this over two data sets: a popular benchmark ontology LUBM, and a synthetic ontology LUBMMOD. LUBM benchmark contains axioms describing simple concept and role hierarchies, role transitivity and inverses. We manually created the LUBMMOD ontology by augmenting LUBM with new axioms that add disjunctions and negations to concept hierarchies.

3. We demonstrate the benefits of implementing an interpolant enumeration procedure on a specialized DL reasoner by contrasting interpolation in FOL with simulation of interpolation on a DL dialect that is carried out on a specialized DL reasoner.

1.2 Thesis Organization

The reminder of the thesis is organized as follows. In Chapter 2, we present the definitions and discuss topics that are needed in order to appreciate this work, in particular description logics, assertion retrieval, definability, interpolation and their applications. We review work in the related areas of instance retrieval, query rewriting and interpolation in Chapter 3. In Chapter 4 we describe our approach theoretically and point out the complications that we had to resolve for our evaluation, as well as modifications that we had to implement in order to resolve those problems. We present a discussion of results of an experimental evaluation in Chapter 5. Finally, conclusion and directions for future work are given in Chapter 6. Definitions and details about first order predicate logic are provided in Appendix A.

Chapter 2

Preliminaries

In this chapter, we provide the necessary background knowledge. We will start by presenting a discussion on description logics, then, we will provide the necessary details about *assertion retrieval algebra*, and finally, we will present necessary definitions for *Beth definability* [11] and *Craig interpolation* [16] and discuss possible application of these concepts to answering instance retrieval queries in first order logic and description logics.

2.1 Description Logic

Description Logics (DL) are a family of knowledge representation formalisms used to represent the basic terminology and facts in an application domain [9]. DLs differ from many other knowledge representation formalisms by having a formal semantics grounded in FOL. Also, description logics allow systems to perform various types of reasoning from knowledge explicitly represented and stored in a knowledge base, to infer facts that are implicitly represented.

A DL knowledge base, $\mathcal{K}$, stores facts and rules that can be used for reasoning in the application domain (i.e. it is an extension of the database). $\mathcal{K}$ consists of two parts: a TBox ($\mathcal{T}$), also called a *terminology* and an ABox ($\mathcal{A}$). Essentially, TBox contains the constraints that define an ontology of the application domain (i.e. intentional knowledge), while ABox stores the data of the application domain (i.e. extensional knowledge).

A TBox introduces the *vocabulary* of the application domain. The vocabulary consists of symbols denoting *concepts*, *individuals*, and *roles* denoting binary relationships between individuals. Using these concept and role names from the vocabulary, more complicated concept descriptions can be formed. The TBox consists of a set of *axioms*, defining relationships between concepts and roles. Subsumption relationship between concepts and roles is written as $C \sqsubseteq D$ and $R \sqsubseteq Q$, respectively, and equivalence relationship between concepts and roles is written as $C \equiv D$ and $R \equiv Q$, respectively.

An ABox consists of a set of assertions about individuals in the application domain, written as $a : C$ or as $(a, b) : R$, stating that individual a belongs to the extension of concept C and a tuple $\langle a, b \rangle$ belongs to the extension of the role R, respectively.

C, D	$::=$	A	(atomic concept)
	$\mid$	$\top$	(top concept)
	$\mid$	$\bot$	(bottom concept)
	$\mid$	$\neg A$	(atomic negation)
	$\mid$	$C \sqcap D$	(conjunction)
	$\mid$	$\forall R.C$	(universal restriction)
	$\mid$	$\exists R.\top$	(limited existential quantification)

Table 2.1: Grammar for $\mathcal{AL}$ concepts.

Depending on which constructors are available for concept, role and axiom definitions, one can form many different DL dialects, and by extension languages $\mathcal{L}$ based on these dialects. The majority of DL dialects are fragments of FOL. The most basic description logic dialect is $\mathcal{AL}$ (defined in Table 2.1). By adding more constructs to $\mathcal{AL}$ we can produce DL dialects with more expressive power. However, by doing this, we may have to pay with higher complexity for reasoning tasks. Generally, a compromise between expressiveness and efficiency is the most important issue when choosing a DL dialect for one's application.

Table 2.1 shows the grammar for general concepts in $\mathcal{AL}$ DL dialect. An $\mathcal{AL}$ TBox contains only axioms of the type $C \sqsubseteq D$ or $C \equiv D$, where C, D are general $\mathcal{AL}$ concepts.

Semantics of $\mathcal{AL}$ can be defined through an *interpretation* $\mathcal{I}$. Similar to FOL, an interpretation consists of two parts: a non-empty set $\triangle^{\mathcal{I}}$ called a *domain*, and a total function $(\cdot)^{\mathcal{I}}$ which maps every atomic concept A to a set $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$, and every atomic role R to a set of tuples $R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$.

An interpretation function is extended to more complex concepts in the following way:

$$\begin{aligned}
\top^{\mathcal{I}} &= \triangle^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \triangle^{\mathcal{I}} - A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \triangle^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} &= \{a \in \triangle^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}
\end{aligned}$$

An interpretation also extends to the TBox axioms: we say that concept C is subsumed by D (i.e. $C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and that C and D are equivalent (i.e $C \equiv D$), if $C^{\mathcal{I}} = D^{\mathcal{I}}$.

Construct Name	Symbol	Syntax	Semantics $((\cdot)^{\mathcal{I}})$
Disjunction	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Full Existential Quantification	$\mathcal{E}$	$\exists R.C$	$\{a \in \triangle^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
Number Restrictions	$\mathcal{N}$	$\geq nR$	$\{a \in \triangle^{\mathcal{I}} \mid \lvert\{b \lvert (a,b) \in R^{\mathcal{I}}\}\rvert \geq n\}$
		$\leq nR$	$\{a \in \triangle^{\mathcal{I}} \mid \lvert\{b \lvert (a,b) \in R^{\mathcal{I}}\}\rvert \leq n\}$
Negation	$\mathcal{C}$	$\neg C$	$\triangle^{\mathcal{I}} - C^{\mathcal{I}}$
Role Inverse	$\mathcal{I}$	R^{-}	$\{(a,b) \in \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}} \mid (b,a) \in R^{\mathcal{I}}\}$
Role Hierarchies	$\mathcal{H}$	$R \sqsubseteq Q$	$R^{\mathcal{I}} \subseteq Q^{\mathcal{I}}$
Transitive Roles	$\mathcal{S}$	$Trans(R)$	$S(a,b) \wedge S(b,c) \rightarrow S(a,c)$
Nominals	$\mathcal{O}$	$\{a_1, \ldots, a_n\}$	$\{a_1^{\mathcal{I}}\} \cup \cdots \cup \{a_n^{\mathcal{I}}\}$

Table 2.2: Description Logic Constructors.

Table 2.2 shows possible constructs for extending the $\mathcal{AL}$ dialect. In Table 2.2, we assume that $a, b, c \in \triangle^{\mathcal{I}}$, C, D are general concepts, R, Q are atomic roles, S is a role R or its inverse R^{-}, and n is a non-negative integer.

Furthermore, we can extend $\mathcal{AL}$ with a construct for a *concrete domain* $\mathbf{D}$. Syntactically, this adds a possibility for saying things like: $f = g$ or $f < k$, where f, g are *features* and k is a constant in the concrete domain. Semantically, our interpretation is extended by $\mathbf{D}^{\mathcal{I}}$ a disjoint concrete domain of finite strings, and interpretation function $(\cdot)^{\mathcal{I}}$ is extended with a mapping of each feature f to a total function $(f)^{\mathcal{I}} : \triangle \rightarrow \mathbf{D}^{\mathcal{I}}$, the "=" symbol to the equality relation over $\mathbf{D}^{\mathcal{I}}$, the "<" symbol to the binary relation for an alphabetic ordering of $\mathbf{D}^{\mathcal{I}}$, and a finite string k to itself.

Using the constructs defined in Table 2.2 and concrete domain construct $\mathbf{D}$, we can create any of the following DL dialects (constructs mentioned in square brackets are optional):

$$\mathcal{AL}[\mathcal{U}][\mathcal{E}][\mathcal{N}][\mathcal{C}][\mathcal{H}][\mathcal{I}][\mathcal{O}][\mathcal{S}][(\mathbf{D})]$$

Often, in the literature, a dialect $\mathcal{ALC}$ with transitive roles $\mathcal{S}$ is abbreviated with a single character — $\mathcal{S}$. This is the convention that we use in this work.

Most things in description logics are inherited from the FOL: *satisfiability*, *logical inference*, *model*, etc. A *signature* of a knowledge base $\mathcal{K}$, written $sig(\mathcal{K})$, is a set of concept and role symbols that appear in $\mathcal{K}$; similarly, a signature of a concept C, written $sig(C)$ is a set of all concept and role symbols that appear in the definition of C.

Suppose, we are given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ over some DL dialect $\mathcal{L}$. As with FOL, there are many standard reasoning problems over DL knowledge bases:

- *Concept satisfiability* accepts a concept description C in $\mathcal{L}$ as input query, and determines whether $\mathcal{K} \models C$.
- *Knowledge base satisfiability*, determines whether there is an interpretation $\mathcal{I}$ that is a model of $\mathcal{K}$.
- *Instance checking* accepts a query concept C and an individual a as input, and determines whether $\mathcal{K} \models a : C$.
- *Instance retrieval* is a problem that is of particular importance to this work. A query for an instance retrieval problem is a concept C in $\mathcal{L}$. An answer to an instance retrieval problem would be a set of individuals: $\{a \mid \mathcal{K} \models a : C\}$.

- *Conjunctive queries (CQ)* is an extension of instance retrieval. We are given a query q with distinguished variables $\overrightarrow{x} = \langle x_1, \ldots, x_n \rangle$. An answer to the CQ is a set of all tuples $\overrightarrow{c} = \langle c_1, \ldots, c_n \rangle$, where each individual c_i appears in $\mathcal{A}$, such that for every interpretation $\mathcal{I}$ that is a model of $\mathcal{K}$, we have $\overrightarrow{c} \in q^{\mathcal{I}}$. The following is a simple, illustrative example of CQ: let the knowledge base be $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T} = \emptyset$, and $\mathcal{A} = \{\text{Tom: } Person,\ \text{UW: } University,\ \langle \text{Tom, UW} \rangle : WorksFor\}$. We can ask the following CQ over this $\mathcal{K}$:

  $$q = Person(x) \sqcap WorksFor(x, y) \sqcap University(y)$$

  Essentially, this query is asking for all employees, together with the organizations where they work. Query (distinguished) variables are $\overrightarrow{x} = \langle x, y \rangle$. Set of answers for q, with respect to a given $\mathcal{K}$ will contain a single tuple: $\{\langle \text{Tom, UW} \rangle\}$. A further, trivial, extension to CQ is a *union of conjunctive queries (UCQ)*, where the setup is the same as for CQ, except that a query is of the form: $q = q_1 \sqcup \cdots \sqcup q_n$, where each q_i is a CQ and where the distinguished variables of any pair q_i and q_j are the same.

Complexity for performing reasoning tasks specified above can be expressed in terms of *data complexity*, measured in terms of the size of a database (or, in case of DLs, the size of ABox), *expression complexity*, measured in terms of the size of the query, and ontology (i.e. TBox), or *combined complexity*, measured in terms of the combined size of $\mathcal{K}$ and the query [59].

This work is largely based on the $\mathcal{SHI}$ DL dialect. A $\mathcal{SHI}$ TBox, in addition to concept inclusion and equivalence axioms, can also contain role inclusion/equivalence axioms as well as transitivity axioms: $Trans(S)$, where S is a role or an inverse of a role. A role S is called *complex* if $\texttt{Trans}(S')$ for some $S' \sqsubseteq^{*} S$, where $\sqsubseteq^{*}$ is a transitive-reflexive closure of $\sqsubseteq$

over the set $\{S_1 \sqsubseteq S_2\} \cup \{S_1^- \sqsubseteq S_2^- \mid S_1 \sqsubseteq S_2\}$. To avoid undecidability [36], a complex role S may occur only in concept descriptions of the form $\neg\exists S.C_1$ or of the form $\exists S.C_1$.

The *DL-Lite* is a family of *light-weight* description logics that is of a particular interest [14]. Concept descriptions in DL-Lite conform to the following grammar:

$$B ::= A \mid \exists R.\top \mid \exists R^-.\top$$
$$C ::= B \mid \neg B \mid C_1 \sqcap C_2$$

Interpretation for these constructs is as defined previously in this chapter. ABox contains assertions of the type $a : B$ and $(a, b) : R$. TBox can contain axioms of the type: $C \sqsubseteq D$, where C, D are general concepts, and $funct(R)$, which states that R is a functional role, meaning that for every individual a, such that $\{a\} \in \exists R.\top$, there exists exactly one individual b that is a filler for a in R. The importance of DL-Lite is due to expressivity (enough to express some benchmark ontologies), and good performance of reasoning: satisfiability of DL-Lite knowledge base $\mathcal{K}$ can be decided in polytime in the size of $\mathcal{K}$, and answering conjunctive queries can be done in PTIME in data complexity (NP-complete in combined complexity).

2.2 Assertion Retrieval

Definitions and discussion in this section derive from work in [62]; concepts introduced below can be adopted to any DL dialect $\mathcal{L}$, however, in this work, we concentrate on the $\mathcal{SHI}$ DL languages.

Suppose an information system is organized in terms of the knowledge base $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$ in the $\mathcal{SHI}$ DL dialect. We refer to named individuals appearing in ABox as *objects*. Answering object queries is represented by instance retrieval.

Definition 1 (Instance Retrieval) *Let $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$ be a knowledge base over $\mathcal{SHI}$ DL dialect. An* instance retrieval *query is represented by a concept C. Answering* instance retrieval *query amounts to computing all named individuals a that occur in $\mathcal{A}$, for which $\mathcal{K} \models a : C$.*

Query concept C in Definition 1 corresponds to a selection condition in relational queries. Object queries can be further generalized, by adding a *projection operation* (a generalization of the relational projection).

Definition 2 (Projection Description) *Projection description Pd, is defined with the following grammar:*

$$Pd \ ::= \ C? \ \mid \ Pd_1 \sqcap Pd_2 \ \mid \ \exists R.Pd$$

where C is an arbitrary concept and R is a role, in the $\mathcal{SHI}$ dialect. A set of concepts $\mathcal{L}_{Pd}$, defined by Pd, is given as follows:

$$\begin{aligned}
\mathcal{L}_{Pd} &::= \{\sqcap S \ \mid \ S \subseteq_{\text{fin}} \mathcal{L}^{TUP}_{Pd}\} \\
\mathcal{L}^{TUP}_{C?} &::= \{C, \top\} \\
\mathcal{L}^{TUP}_{Pd_1 \sqcap Pd_2} &::= \{C_1 \sqcap C_2 \mid C_1 \in \mathcal{L}^{TUP}_{Pd_1}, C_2 \in \mathcal{L}^{TUP}_{Pd_2}\} \\
\mathcal{L}^{TUP}_{\exists R.Pd_1} &::= \{\exists R.C \ \mid \ C \in \mathcal{L}_{Pd_1}\}
\end{aligned}$$

where $\subseteq_{\text{fin}}$ is a finite subset. Let S be the set of concepts specified by $\mathcal{L}_{Pd}$ and $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a $\mathcal{SHI}$ knowledge base. We denote the most specific *concepts with respect to $\mathcal{L}_{Pd}$ as:*

$$\lfloor S \rfloor_{\mathcal{K}} \ = \ \{C \in S \ \mid \ \textit{there does not exist } D \in S : (\mathcal{K} \models D \sqsubseteq C, \mathcal{K} \not\models C \sqsubseteq D)\}$$

This will transform the instance retrieval problem into an *assertion retrieval* problem. In the context of assertion retrieval, a user query is a pair (C, Pd), where C is a $\mathcal{SHI}$

concept (same as in Definition 1), and Pd defines a special subset $\mathcal{L}_{Pd}$ of concepts in $\mathcal{SHI}$ (corresponding to projection in relational setting), as per Definition 2. To illustrate the definition above, suppose we have $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T} = \{A_1 \sqsubseteq \exists R.A_2\}$ and $Pd = A_1? \sqcap \exists R.A_2?$. Then,

$$\mathcal{L}_{Pd} = \{\sqcap S' \mid S' \subseteq_{fin} \{(\top \sqcap \exists R.A_2), (\top \sqcap \exists R.\top), (A_1 \sqcap \exists R.\top), (A_1 \sqcap \exists R.A_2)\}\}$$

Let $S = \{D \in \mathcal{L}_{Pd} \mid \mathcal{K} \models A_1 \sqsubseteq D\}$, then we get $\lfloor S \rfloor_{\mathcal{K}} = \{A_1 \sqcap \top, A_1 \sqcap \exists R.A_2\}$, from which we further derive $\lfloor \lfloor S \rfloor_{\mathcal{K}} \rfloor_{\emptyset} = \{A_1 \sqcap \exists R.A_2\}$, and finally get $[\![S]\!]_{\mathcal{K}} = A_1 \sqcap \exists R.A_2$.

Answering an assertion retrieval query amounts to reporting all assertions $a : C_a$ such that $\mathcal{K} \models a : C \sqcap C_a$, where C_a is the most specific concept with respect to $\mathcal{L}_{Pd}$, for which this logical consequence holds.

For notation convenience, denote the *minimum concept* in the set of concepts $\lfloor \lfloor S \rfloor_{\mathcal{K}} \rfloor_{\emptyset}$ (according to some arbitrary total ordering) by $[\![S]\!]_{\mathcal{K}}$. Now, we can define more formally the query semantics for assertion retrieval problem.

Definition 3 (Assertion Retrieval Query Semantics) *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ knowledge base. A user query $Q = (C, Pd)$ over $\mathcal{K}$ computes the following set of assertions:*

$$\{a : [\![\{D \mid D \in \mathcal{L}_{Pd}, \mathcal{K} \models a : D\}]\!]_{\mathcal{K}} \mid \mathcal{K} \models a : C, a \text{ occurs in } \mathcal{K}\}$$

Now, we can define *assertion algebra*, to manipulate assertion retrieval query concepts. Assertion algebra allows expressing queries in terms of cached results.

Definition 4 (Cached Query Result) *Cached query result S_i is a set of concept assertions computed by user query (C_i, Pd_i), with respect to the same $\mathcal{SHI}$ knowledge base $\mathcal{K}$.*

Definition 5 (Assertion Algebra) *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ knowledge base. Assertion algebra contains all of the operators defined by the following grammar.*

$$
\begin{aligned}
Q ::= C \quad & \{a : C \mid a \text{ occurs in } \mathcal{K}\} \\
\mid \; P^{\mathcal{K}} \quad & \{a : \top \mid a \text{ occurs in } \mathcal{K}\} \\
\mid \; S_i(Q) \quad & \{a : C \mid (a : C) \in S_i, (a : D) \in Q, \{a : C\} \models a : D\} \\
\mid \; \sigma_C^{\mathcal{K}}(Q) \quad & \{a : D \mid (a : D) \in Q, \mathcal{K} \cup \{a : D\} \models a : C\} \\
\mid \; \pi_{Pd}^{\mathcal{K}}(Q) \quad & \{a : [\![\{D \mid \mathcal{K} \cup \{a : C\} \models a : D, D \in \mathcal{L}_{Pd}\}]\!]_{\mathcal{K}} \mid (a : C) \in Q\} \\
\mid \; Q_1 \cap Q_2 \quad & \{a : D_1 \sqcap D_2 \mid (a : D_i) \in Q_i, i = 1, 2\}
\end{aligned}
$$

where C is a general $\mathcal{SHI}$ concept and S_i is a cached query result.

Using assertion algebra, a user query $Q = (C, Pd)$, with respect to some $\mathcal{SHI}$ knowledge base $\mathcal{K}$, can be rewritten into an equivalent assertion algebra expression: $Q' = \pi_{Pd}^{\mathcal{K}}(\sigma_C^{\mathcal{K}}(P^{\mathcal{K}}))$.

Finally, we link assertion algebra expressions with sets of concepts in the $\mathcal{SHI}$ DL dialect, with the following construction:

Definition 6 (Representative Language for Algebraic Expressions) *$\mathcal{L}_Q$ (defined below) is a language representing concepts of query Q expressed in assertion algebra.*

$$
\mathcal{L}_Q = \begin{cases}
\{C\} & if\ Q = \text{"}C\text{"}; \\
\{\top\} & if\ Q = \text{"}P^{\mathcal{K}}\text{"}; \\
\mathcal{L}_{Pd_i} & if\ Q = \text{"}S_i(Q_1)\text{"}; \\
\mathcal{L}_{C \cap Q_1} & if\ Q = \text{"}\sigma_C^{\mathcal{K}}(Q_1)\text{"}; \\
\mathcal{L}_{Pd} & if\ Q = \text{"}\pi_{Pd}^{\mathcal{K}}(Q_1)\text{"}; \\
\{C \sqcap D \mid C \in \mathcal{L}_{Q_1}, D \in \mathcal{L}_{Q_2}\} & if\ Q = \text{"}Q_1 \cap Q_2\text{"};
\end{cases} \tag{2.1}
$$

where Pd_i is the projection description used to compute cached result S_i.

2.3 Beth Definability and Interpolation

In this section, we will present the definitions and discussion on Beth definability and Craig interpolation for first order logic, and then extend the discussion to the DL setting.

Definition 7 (Implicit Definability) *Let Σ be a first order theory and $S \subseteq sig(\Sigma)$ a set of atomic predicates. A first order formula ϕ with $FV(\phi) = \langle x_1, \ldots, x_n \rangle$ is* implicitly definable *from S under Σ if $\phi^{\mathcal{I}_1} = \phi^{\mathcal{I}_2}$ for any two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, for which the following conditions hold:*

- $\triangle^{\mathcal{I}_1} = \triangle^{\mathcal{I}_2}$
- $\mathcal{I}_1 \models \Sigma$ *and* $\mathcal{I}_2 \models \Sigma$.
- $P^{\mathcal{I}_1} = P^{\mathcal{I}_2}$ *for every* $P \in S$.

Definition 8 (Explicit Definability) *Let Σ be a first order theory and $S \subseteq sig(\Sigma)$ a set of atomic predicates. A first order formula ϕ with $FV(\phi) = \langle x_1, \ldots, x_n \rangle$ is* explicitly definable *from S under Σ, if there exists another well formed formula ψ with $FV(\psi) = \langle x_1, \ldots, x_n \rangle$ and $sig(\psi) \subseteq S$, such that:*

$$\Sigma \models \forall x_1, \ldots, x_n (\phi \equiv \psi)$$

The formula ψ is called an explicit definition *of ϕ from S with respect to Σ.*

In essence, a statement that a formula ϕ is implicitly definable from a set S under Σ means that the set S contains enough information to uniquely determine the interpretation

(or extension) of ϕ with respect to Σ. Explicit definability takes this one step further: for a formula ϕ explicitly definable from S under Σ, not only does S contain enough information to uniquely identify ϕ, S contains enough information to write down another formula ψ with exactly the same interpretation as ϕ, such that the signature of ψ contains only predicates that appear in S.

Definition 9 (Beth Definability Property) *Suppose Σ is a first order theory and $S \subseteq sig(\Sigma)$ a set of atomic predicates.* Beth definability *is a property that states: any formula ϕ that is implicitly definable from S under Σ is also explicitly definable from S under Σ.*

For the reminder of this work, we will use terms Beth definability and definability interchangeably. Beth definability is a property of a logic; some logics do not have this property. Definition 9 states that for a logic with the definability property (like FOL), an explicit definition for a formula can always be found, given that the formula is implicitly definable, however, it does not provide a method for finding explicit definitions. One possible way of producing explicit definitions is through Craig interpolation.

Definition 10 (Craig's Interpolation) *Suppose ϕ_1 and ϕ_2 are two well formed formulas in FOL, and that $\models \phi_1 \rightarrow \phi_2$. Then, there exists another well formed formula ψ, such that $sig(\psi) \subseteq sig(\phi_1) \cap sig(\phi_2)$, and $\models \phi_1 \rightarrow \psi \rightarrow \phi_2$. This well formed formula ψ is called an* interpolant*.*

The definition of Craig's interpolant could be trivially extended to be with respect to some first order theory Σ. There are constructive procedures for generating interpolants from the tableaux proof of $\models \phi_1 \rightarrow \phi_2$ (these procedures can also be generalized to other proof systems, like resolution-refutation, etc.) [18]. For first order logic interpolant extraction is semi-decidable.

Implicit/Explicit definability and interpolation definitions can also be very easily extended to the DL setting. The following definitions assume some arbitrary DL dialect $\mathcal{L}$, a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ over $\mathcal{L}$.

Definition 11 (DL — Implicit Definability) *Let C be a concept in $\mathcal{L}$, and a set $\Sigma \subseteq sig(C, \mathcal{T})$. We say that concept C is* implicitly definable *from Σ under $\mathcal{T}$ if and only if for any two models $\mathcal{I}_1$ and $\mathcal{I}_2$ of $\mathcal{T}$, if:*

- $\triangle^{\mathcal{I}_1} = \triangle^{\mathcal{I}_2}$
- *for all $P \in \Sigma$, $P^{\mathcal{I}_1} = P^{\mathcal{I}_2}$*

then $C^{\mathcal{I}_1} = C^{\mathcal{I}_2}$

Definition 12 (DL — Explicit Definability) *Let C be a concept in $\mathcal{L}$, and $\Sigma \subseteq sig(C, \mathcal{T})$. C is* explicitly definable *from Σ under $\mathcal{T}$ if and only if there is some concept D in $\mathcal{L}$ such that $\mathcal{T} \models C \equiv D$ and $sig(D) \subseteq \Sigma$. We call such a concept D* explicit definition *of C from Σ under $\mathcal{T}$.*

Definition 13 (DL — Beth Definability Property) *A description logic dialect $\mathcal{L}$ has* Beth definability property*, if for any concept C in $\mathcal{L}$, TBox $\mathcal{T}$ over $\mathcal{L}$ and any $\Sigma \subseteq sig(C, \mathcal{T})$, if C is implicitly definable from Σ under $\mathcal{T}$, then C is also explicitly definable from Σ under $\mathcal{T}$.*

Definition 14 (DL — Interpolant) *Let C and D be concepts in description logic $\mathcal{L}$, and $\mathcal{T}_1$, $\mathcal{T}_2$ be two TBoxes in the same description logic, such that $\mathcal{T}_1 \cup \mathcal{T}_2 \models C \sqsubseteq D$. A concept I in $\mathcal{L}$ is called an interpolant of C and D under $\mathcal{T}_1 \cup \mathcal{T}_2$ if all of the following conditions hold:*

- $sig(I) \subseteq sig(C, \mathcal{T}_1) \cap sig(D, \mathcal{T}_2)$
- $\mathcal{T}_1 \cup \mathcal{T}_2 \models C \sqsubseteq I$
- $\mathcal{T}_1 \cup \mathcal{T}_2 \models I \sqsubseteq D$

Unfortunately, not all DL dialects possess definability property. In [55] and [56], authors discuss and summarize definability property, and weaker version of definability — *concept name Beth definability property (CBP)*, for various decidable, expressive DL dialects. CBP enforces all role names to appear in the set of concepts from which explicit definitions can be generated. Also, in [56], authors give a constructive algorithm for building interpolants from DL tableaux. Although not all decidable DL dialects possess the definability property, for those that do, interpolant generation is also decidable.

In the next section, we elaborate on the relation between definability and interpolation, and discuss the importance of the definability property for query answering in FOL, with trivial extension to DLs.

2.4 Definability and Interpolation in Query Evaluation

As described in Appendix A, first order formulas can be perceived as user queries, with free variables as query variables. Let Σ be a first order theory that contains only domain independent formulas, Q — a first order query, and $S \subset sig(\Sigma)$ a set of first order atomic predicates; one can think of S as a set of *materialized views* (i.e. stored results of evaluating past user queries). To answer user query Q, we need to rewrite it as Q_p, so that $sig(Q_p) \subseteq$

$sig(S)$ and $\Sigma \models Q \equiv Q_p$. If this rewriting is executable (i.e. translatable into a computer program in some programming language) and efficient, we call Q_p a *query plan.*

Beth definability and Craig interpolation allow us to generate a plan Q_p for user query Q. We generate a new theory Σ^* by inspecting every $\phi \in \Sigma$, and replacing every predicate P that appears in ϕ such that $P \notin S$, with a new predicate symbol P^*, and generate a new query Q^* from Q, by the same procedure. Then, we can say that query Q is definable if and only if:

$$(\Sigma \cup \Sigma^*) \models (Q \rightarrow Q^*)$$

which can be reformulated as an interpolant generation problem:

$$\models ((\wedge\Sigma) \wedge Q) \rightarrow ((\wedge\Sigma^*) \rightarrow Q^*)$$

The generated interpolant Q_p will have the properties that $sig(Q_p) \subseteq sig(S)$ and $\Sigma \models Q \equiv Q_p$, which means that rewriting Q_p is a candidate query plan for Q.

Weddell and Toman in [57] describe a procedure for enumerating different interpolants, by enumerating finite tableau proofs of $(\wedge\Sigma) \wedge (\wedge\Sigma^*) \wedge Q \wedge (\neg Q^*)$. There are two reasons why enumerating interpolants is needed for query evaluation: to generate executable interpolants, and to generate efficient interpolants.

In some cases, a generated rewriting Q_p cannot be turned into an executable plan. In those circumstances, the system would ask for further interpolants, until an executable one is found. Discussion of why interpolants may not generate an executable query plan is outside the scope of this thesis. We will only note that for this work, interpolants are not required to be executable.

A query plan must also be reasonably efficient. To satisfy this requirement, we may have to enumerate multiple interpolants, and choose the most efficient one according to some cost model.

The discussion above is trivially extended to DL dialects. Suppose $\mathcal{L}$ is a description logic dialect with definability property ([56]). Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, and C be an instance query in $\mathcal{L}$. Let S be a set of concepts in $\mathcal{L}$, such that $sig(S) \subseteq sig(\mathcal{T})$, without loss of generality we can assume that S contains only atomic concepts and roles. Same as above, we need to find a rewriting C_p of C, such that $\mathcal{T} \models C \equiv C_p$, $sig(C_p) \subseteq sig(S)$, and C_p is executable and sufficiently efficient according to some evaluation metric.

Using the same process as in FOL case, we can create a new TBox $\mathcal{T}^*$ and a query C^*. By running interpolant extraction procedure [56], we can find a rewriting C_p such that $sig(C_p) \subseteq sig(S)$ and $\models (\sqcap\mathcal{T}) \sqcap (\sqcap\mathcal{T}^*) \sqcap C \sqcap (\neg C^*)$, which by the virtue of construction of $\mathcal{T}^*$ and C^* means that $\mathcal{T} \models C \equiv C_p$. Finally, interpolant enumeration procedure can be extended to description logic, to produce efficient and executable interpolant C_p.

Chapter 3

Related Work

Our work addresses a problem of optimizing the plan generation step for answering assertion retrieval queries over a $\mathcal{SHI}$ knowledge base [46]. Our approach uses definability and interpolation to produce a suitable rewriting of the original query, so that reasoning with respect to a knowledge base can be discarded for the selection operator of the query. Thus, we touch on a few separate areas of related research. Our approach is tightly integrated with description logics and first order logic as frameworks for writing ontologies, performing reasoning, answering queries and performing interpolant enumeration. A concise overview of first order predicate logic is presented in Appendix A.

Also, our work touches on the problem of compiling instance queries and conjunctive queries into executable query plans, and evaluating those plans to get the set of results. Generating executable plans for query answering has been a central issue in databases, and many other fields that are tightly coupled with information systems. Relational databases have dealt with query compilation since their emergence [51]. This problem remains important with the OBDA and other information systems that use DL knowledge bases. An-

swering instance retrieval or conjunctive queries over DL knowledge base $\mathcal{K}$ adds another complication to the task — reasoning with respect to $\mathcal{K}$. Due to this, many reasoning techniques had to be adapted specifically to a DL environment [10, 24]. Complexity of conjunctive query answering over relational databases is in AC^0 [5] complexity class. This task becomes significantly harder when expressive DL knowledge bases are considered. In particular, for some expressive DLs, complexity of conjunctive query answering is co-NP complete in the size of the data [37, 42]. This served as a reason for many optimization techniques and heuristics for DL instance retrieval [30], and development of the light-weight DL families, like DL-Lite, where conjunctive query answering can be done in PTIME data complexity [14].

Finally, our work is tightly related to the topics of query rewriting, definability and interpolation. More details on these areas are provided in the next two sections.

3.1 Query Rewriting

Query rewriting is a very important technique for query compilation and answering. It has been studied extensively for decades in various contexts: relational databases, DL knowledge bases, OBDA, etc. Query rewriting has been studied from multiple angles: as a distinct optimization step in the query compilation procedure, as an abstraction of the entire query compilation process, and finally, as something in between of these two extremes.

In relational databases, query rewriting is used in two main directions: to optimize plan generation, and to answer the original user query over the set of views. When query rewriting is used as an optimization step in plan generation, its main goal is to convert

the input query into an equivalent query, in the hope that it will be easier for the optimizer to process the new query. The rewriting attempts to eliminate unneeded operators (for example duplicate elimination [44]), and selection conditions that are always satisfied or that are implied by other selection conditions [60]. Also, query rewriting in relational databases is applied to simplify and optimize the execution of nested queries and aggregation operations [22, 6]. There is a large variety of query rewriting techniques and heuristics used in commercial relational database systems that remain a trade secret, and thus are unpublished. Another reason to use query rewriting in relational databases is to attempt to answer an original user query with the set of available views [31]. Here, the task of rewriting is broken up into two parts: determining whether the set of available views is sufficient to answer a query, and producing the actual rewriting of a user query in terms of views; both of these are very hard problems. More details on the current state-of-the-art in this area can be found in [7].

Another way to look at query rewriting is to view query compilation as a rewriting of the original user query into an executable query plan. This is the approach adopted in [57]. In this approach, a user query is issued over a logical view of the data, and the purpose of query compilation is to produce another query, over the physical view of the data (i.e. over access paths for the data [51]), with conditions that the new query is executable and "reasonably" efficient with respect to some cost metric.

Another important use of query rewriting is for answering conjunctive queries in DL-Lite. Usage of query rewriting in the DL-Lite case falls in between an optimization step of query compilation and abstracting the entire query compilation process. At the base of query rewriting for DL-Lite is *first-order rewritability (FO-rewritability)*, which, essentially, is a property of a language that given a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, allows to rewrite a query Q into another query Q', such that evaluating Q' on $\mathcal{A}$ returns the same answers

as evaluating Q with respect to $\mathcal{K}$. To evaluate a conjunctive query Q over a DL-Lite knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where the ABox $\mathcal{A}$ is represented as a relational database, we would process each atom of Q using $\mathcal{T}$, and produce a union of conjunctive queries Q', which is a first order expression that can be evaluated directly over the relational database storing $\mathcal{A}$ [14]. Various extensions and modifications of this procedure include adoption of the approach to other DL languages that enjoy FO-rewritability property [12], and extension of the procedure to work with various integrity constraints defined over a DL-Lite knowledge base $\mathcal{K}$ in an attempt to reduce the size of the resulting rewriting Q' [48, 49].

Finally, Franconi, Kerhet and Ngo, in a series of papers [19, 20, 21], applied definability and interpolation to query rewriting. In this scenario, user queries are issued over a first order logic ontology defined over a relational database. The authors use definability and interpolation in FOL to characterize the precise conditions under which it is possible to rewrite the original query as a range restricted first order formula over the relational database, and propose a method based on interpolant extraction from tableau proof [18] to produce the mentioned rewriting. In [21], their approach is extended to ontologies over the $\mathcal{ALCHOIQ}$ DL dialect.

3.2 Definability and Interpolation

Work on definability and interpolation originated in the 50's due to research of Beth [11], and Craig [16], respectively. The original results were presented and proved for the general case of FOL. From that time, much research has been directed to extending the principles to other logics [50, 23, 13, 39]. Various application scenarios were studied for definability and interpolation, for instance to model checking [41] and to query rewriting.

Extending definability and interpolation to description logics has also been studied extensively. A lot of effort has been directed towards describing the Beth definability property for various DL dialects and classifying DLs based on the presence of definability property [55, 56, 54]. In the same publications, the authors investigated bounds on the size of explicit definitions that can be generated for various DL dialects. Finally, in [56], an interpolant extraction procedure for description logics was proposed. Also, various modifications to the original definitions of definability were investigated: a weaker version (CBP) of Beth definability in [55], and a stronger version, called *projective Beth definability*, was investigated in [34]. Applications of definability and interpolation in description logics has also been investigated extensively [9].

Over the past few years, the application of definability and interpolation to query answering over ontologies has also been studied. For example, in [53, 19, 20, 19], authors used definability and interpolation to translate queries over FOL ontology into SQL queries over relational databases. In this thesis we propose a more general application of definability and interpolation to query compilation, which does not require the data to be stored in the form of relational databases. Finally, in [57], Beth definability and interpolation were applied to generate an executable and reasonably efficient query plan, over an arbitrary ontology in domain independent fragment of first order logic, which is an undecidable problem.

Chapter 4

Procedure for Eliminating Reasoning with Respect to a Knowledge Base

The difficulty of answering instance retrieval queries over DL knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, results from the need to reason with respect to that knowledge base. Complexity of reasoning is proportional to the size and complexity of $\mathcal{K}$. This serves as a motivation of reducing the size and complexity of the knowledge base (especially TBox $\mathcal{T}$) required for answering queries, in particular reducing $\mathcal{K}$ to the empty knowledge base $\emptyset$.

Definition 15 (Rewritably Equivalent Concepts) *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base over the $\mathcal{SHI}$ DL dialect, and $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$ a user query expressed in assertion algebra, where C is a selection concept, and Pd a projection description. We define a concept D to be* rewritably equivalent *to concept C if:*

(i) $\mathcal{T} \models C \equiv D$, *and*

(ii) $D \in \mathcal{L}_{Q_1}$

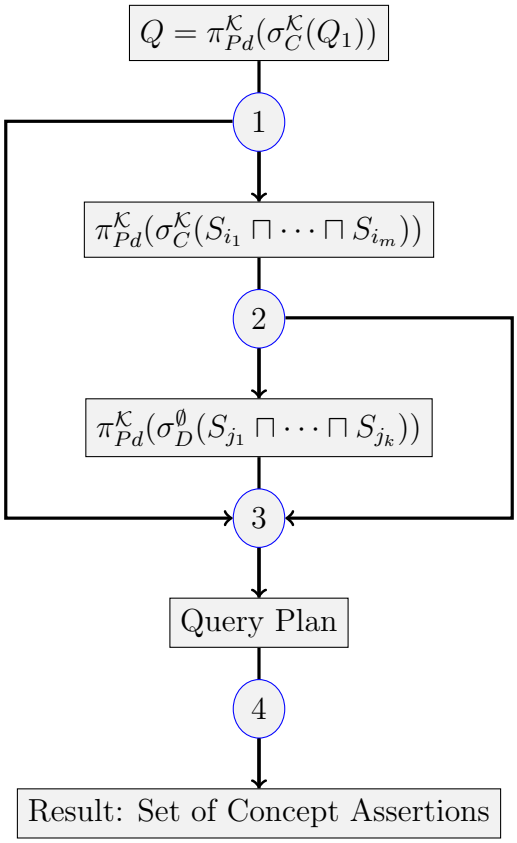

Figure 4.1: Query evaluation steps.

Lemma 1 *Given a $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and a user query $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$, if a concept D is rewritably equivalent to C with respect to $\mathcal{K}$, then a new query $Q' = \pi^{\mathcal{K}}_{Pd}(\sigma^{\emptyset}_{D}(Q_1))$ produces the same set of results as Q.*

Proof: The proof follows immediately from Definition 15.

□

Initially, in the information system, we start with a $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a set of cached query results $S = \{S_1, \ldots, S_n\}$, where each S_i is as per Definition 4. In

this system, users can issue queries of the form $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$, the system will create an efficient executable plan for the query and execute that plan to get the result — a set of concept assertions.

Definition 16 (Relevant Cached Query Results) *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$ a user query, and $S = \{S_1, \ldots, S_n\}$ a set of cached query results, where each $S_i \in S$ is a result of evaluating a query (C_i, Pd_i). We say that $S_i \in S$ is* relevant *for a query Q, if $\mathcal{K} \models C \sqsubseteq C_i$.*

User query Q is evaluated over the knowledge base by following the steps in Figure 4.1. The first step determines whether cached query results, from the set $S = \{S_1, \ldots, S_n\}$, can be utilized to answer the user query. If cached query results are not usable, we can go directly to step 3, with input $\pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$. On the other hand, if cached results can be utilized for answering the query, step one produces a set of relevant cached query results $S' = \{S_{i_1} \ldots S_{i_m}\}$ (as per Definition 16), such that the original query can be rewritten as $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(S_{i_1} \sqcap \cdots \sqcap S_{i_m}))$. Techniques for computing a set of relevant cached results are well known and studied for the case of relational data model [26], and can be extended to our circumstances. Now, we can proceed to step 2, in which we attempt to further rewrite the query. Once again, this may not be possible, in which case we move on to step 3, with input $\pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(S_{i_1} \sqcap \cdots \sqcap S_{i_m}))$. If we succeed in step 2, a query will be rewritten into $\pi^{\mathcal{K}}_{Pd}(\sigma^{\emptyset}_{D}(S_{j_1} \sqcap \cdots \sqcap S_{j_k}))$ (which will be used as an input to step 3), where D is a rewritably equivalent concept to C, and a set of cached results $S'' = \{S_{j_1}, \ldots S_{j_k}\} \subseteq S'$. Note, if this rewriting is possible, than the new selection condition D can be answered from cached query results only. This means that we do not have to reason with respect to the entire $\mathcal{K}$; thus, we can replace $\mathcal{K}$ in the selection condition, by $\emptyset$. In step 3, the system generates an executable query plan (the plan may be optimized in terms of join order, removing

unnecessary operators, etc.), and go on to execute the query plan in step 4, to get the result set of concept assertions.

The procedure for step 2 could be implemented in a brute force way: enumerate through every concept $D_j \in \mathcal{L}_{Q_2}$, where $Q_2 = S_{i_1} \sqcap \cdots \sqcap S_{i_m}$, and check if $\mathcal{T} \models C \sqsubseteq D_j$ and $\mathcal{T} \models D_j \sqsubseteq C$. Assign the new selection concept D to be D_j that passes the above subsumption checks. Finally, we set $S'' = \{D_i \mid D_i \in S' \text{ and } D_i \text{ syntactically occurs in } D\}$. If the entire $\mathcal{L}_{Q_2}$ is exhausted, and no concept $D' \in \mathcal{L}_{Q_2}$ passed the two subsumption checks, we conclude that the query cannot be rewritten in order to eliminate reasoning with respect to $\mathcal{K}$, and we move on to step 3. This method is potentially very slow, since subsumption checks take exponential time, in terms of the size of theory, in the worst case, and the size of $\mathcal{L}_{Q_2}$ grows exponentially as the number of relevant cached query results increases. So, in the worst case, we may have to perform a potentially exponential number of subsumption checks, of which each can take exponential amount of time. Of course, in practice, performance of the brute force method depends on the order of enumeration of concepts in $\mathcal{L}_{Q_2}$, which is impossible to determine without performing reasoning.

In this chapter we concentrate on optimizing the procedure for step 2, by applying the Beth definability property and interpolation. For our approach, we are given a $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an input query $\pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_2))$, where $Q_2 = S_{i_1} \sqcap \cdots \sqcap S_{i_m}$, same as above. A concept D, rewritably equivalent to C, is generated in two steps. First, we enumerate the interpolants D_j of C (for each such D_j, we will have $\mathcal{T} \models D_j \equiv C$). Second, for each interpolant D_j, we check if it syntactically occurs in $\mathcal{L}_{Q_2}$, and if it does — stop the interpolant enumeration, and make the new selection concept D to be D_j. To generate $S'' = \{S_{j_1}, \ldots, S_{j_k}\}$, we follow the same process as for brute force method (described above). If we cannot generate a single interpolant for C, we conclude that it is impossible to rewrite the original query in order to eliminate reasoning with respect to $\mathcal{K}$,

and we move on to step 3.

In Section 2.4, we described how interpolant enumeration can be used to produce a query plan: enumerate interpolants, until we find executable, efficient rewriting for the supplied query. In our optimization, however, we do not need to generate the final query plan, we optimize one of the steps of generating a query plan. For this, we need to generate a rewriting equivalent to the original user query (interpolant extraction takes care of that), but the rewriting does not need to be executable nor efficient. However, enumerating interpolants is still needed for our procedure since the generated interpolants need to conform to certain structural constraints in order to be translated into a $\mathcal{SHI}$ concept, and must also belong to $\mathcal{L}_{Q_2}$. So, for our approach, we essentially adopt the same procedure as in Section 2.4 with different conditions to stop enumerating interpolants.

Our approach is not dependent on any specific DL reasoner, and so it can be implemented as an ad-hoc optimization to any existing DL reasoner. The next two sections describe in detail the two parts of the proposed optimization.

4.1 Assertion Retrieval Algebra to Interpolant Enumeration

First, we need to formulate an interpolant extraction/enumeration problem. This can be done rather easily by following the outline in Section 2.4. Roughly, the TBox $\mathcal{T}$ would be our theory, selection concept C would be our query, and concepts in $\mathcal{L}_{Q_2}$ would be our shared alphabet (i.e. set S as described in Definition 9). A procedure for extracting an interpolant from the DL tableaux is described in [56]. By applying techniques from [57] to this procedure, we can enumerate DL interpolants. However, since we do not have an

implementation of interpolant extraction tool for DLs, we had to use ITB — interpolant enumeration tool implemented for FOL, for the experimental evaluation of the approach. Due to this, the procedure of interpolant enumeration became more complicated.

$\mathcal{SHI}$ Construct	Name	Translate(param1, param2)
$\top$	(TOP)	$True$
A	(Atomic concept)	$A(x)$
S	(Role/Inverse role)	$S(x, y)$
$\neg C$	(General negation)	$\neg Translate(C, x)$
$C \sqcap D$	(Conjunction)	$Translate(C, x) \wedge Translate(D, x)$
$C \sqcup D$	(Disjunction)	$Translate(C, x) \vee Translate(D, x)$
$\exists S.C$	(Existential quantification)	$\exists y S(x, y) \wedge Translate(C, y)$
$\forall S.C$	(Universal restriction)	$\forall y S(x, y) \rightarrow Translate(C, y)$
$C \sqsubseteq D$	(Concept inclusion)	$\forall x(Translate(C, x) \rightarrow Translate(D, x))$
$C \equiv D$	(Concept equivalence)	$\forall x(Translate(C, x) \rightarrow Translate(D, x))$
		$\forall x(Translate(D, x) \rightarrow Translate(C, x))$
R_1 inverse R_2	(Inverse roles)	$\forall x, y R_1(x, y) \rightarrow R_2(y, x)$
		$\forall x, y R_2(x, y) \rightarrow R_1(y, x)$
$S_1 \sqsubseteq S_2$	(Role/Inverse inclusion)	$\forall x, y S_1(x, y) \rightarrow S_2(x, y)$
$S_1 \equiv S_2$	(Role/Inverse equivalence)	$\forall x, y S_1(x, y) \rightarrow S_2(x, y)$
		$\forall x, y S_2(x, y) \rightarrow S_1(x, y)$
$Trans(R)$	(Transitive roles)	$\forall x, y, z((R(x, y) \wedge R(y, z)) \rightarrow R(x, z))$

Table 4.1: $\mathcal{SHI}$ to FOL mapping.

ITB extracts interpolants from the FOL tableaux tree, which means that in the worst case, it may not terminate (due to undecidability of FOL). However, for most practical cases, we found that the tool works rather well. An input to the tool is: a first order theory Σ (i.e. a set of domain independent sentences), a set of first order predicates A_p (i.e. a shared alphabet) and a first order query Q. As our description logic query answering system, we adopted an assertion retrieval engine called $CARE$ [62], implemented for the $\mathcal{SHI}$ DL dialect. Due to the mismatch between languages used in CARE and ITB, we had to implement a translator from $\mathcal{SHI}$ to FOL, and vice versa. Since, in terms of expressive power, $\mathcal{SHI}$ is a strict subset of FOL, the translation from $\mathcal{SHI}$ to FOL can be done without any loss of information. Let $Translate(param1, param2)$ be a function that translates $\mathcal{SHI}$ DL into FOL; first parameter is a concept/role name, and the second parameter is a variable name. Table 4.1 shows mappings from $\mathcal{SHI}$ constructs to FOL.

Each of the inputs to ITB is created by translating a corresponding entity from $\mathcal{SHI}$ into FOL. A first order query Q is obtained by translating a selection concept C to an FOL formula which can be done simply by applying rules from Table 4.1.

A first order theory Σ is constructed from two sources. First, we need to translate every axiom in TBox $\mathcal{T}$ into its FOL equivalent. Once again, this is a rather trivial procedure, that can be done by applying rules from Table 4.1. However, ITB works only with domain independent well formed formulas; due to this, all non-domain independent axioms, if there are any in the TBox, are ignored during translation. In practice, most ontologies can be expressed with domain independent constraints, so working only with domain independent formulas has a rather small effect on the expressiveness of the ontologies. The second type of sentences in Σ comes from computing a set of predicates A_p, described below.

A set of predicates A_p is obtained from $\mathcal{L}_{Q_2}$ (Definition 6), where $Q_2 = S_{i_1} \sqcap \cdots \sqcap S_{i_m}$. We convert $\mathcal{L}_{Q_2}$ into a set of predicates A_p using the translation function from Table 4.1.

Let A' represent a fresh atomic concept name that does not appear anywhere in the knowledge base. We start with $A_p = \emptyset$, and we process each $C \in \mathcal{L}_{Q_2}$, and add appropriate predicates to A_p, and appropriate constraints to Σ. There are four different cases to consider, depending on concept C:

1. $A_p = A_p \cup C(x)$, if C is an atomic concept. No additional constraints need to be added to Σ.

2. $A_p = A_p \cup A'(x)$, if $C := C_1 \sqcap C_2$ or $C := C_1 \sqcup C_2$ or $C := \neg C_1$, where C_1, C_2 are arbitrary $\mathcal{SHI}$ concepts. In this case, we also need to add axioms $\forall x(A'(x) \rightarrow Translate(C, x))$ and $\forall x(Translate(C, x) \rightarrow A'(x))$ to Σ.

3. $A_p = A_p \cup C_1(x) \cup S(x, y)$, if $C := \exists S.C_1$ or $C := \forall S.C_1$ and C_1 is an atomic concept and S is a role or its inverse. No additional constraints need to be added to Σ.

4. A_p $A_p \cup A'(x) \cup S(x, y)$, if $C := \exists S.C_1$ or $C := \forall S.C_1$ and C_1 is an arbitrary $\mathcal{SHI}$ concept and S is a role or its inverse. In this case, we need to add axioms $\forall x(A'(x) \rightarrow Translate(C_1, x))$ and $\forall x(Translate(C_1, x) \rightarrow A'(x))$ to Σ.

Once the above steps are finished, we start ITB tool with Σ, A_p and Q as input, and it generates a stream of well formed formulas $D_1, D_2, \ldots$ (i.e. interpolants), such that $\Sigma \models Q \equiv D_i$. Each D_i is supplied as input to the second step of our approach, described below.

4.2 Interpolant to $\mathcal{SHI}$ Query Concept

Each interpolant D_i that is produced by ITB needs to be converted into $\mathcal{SHI}$ concept. This is a rather difficult task, for two reasons: closure of $\mathcal{SHI}$ dialect under definability

property, and greater expressivity of FOL compared to $\mathcal{SHI}$.

$\mathcal{SHI}$ DL dialect is not closed under Beth definability property. This means that in some cases there is no $\mathcal{SHI}$ concept D which is an interpolant of input $\mathcal{SHI}$ concept C. It may be the case that an interpolant for C exists in another DL dialect that is more expressive than $\mathcal{SHI}$. In our approach, ITB is implemented for domain independent fragment of FOL, which is closed under Beth definability. Therefore, we can be sure that if an interpolant D_i exists for C, it will be found. However, it may be impossible to express D_i as a $\mathcal{SHI}$ concept. This problem can be partially avoided by using a DL dialect other than $\mathcal{SHI}$ for the query answering system, that is closed under definability property (for example by removing role hierarchies from $\mathcal{SHI}$ we get a dialect $\mathcal{ALCIS}$, which is closed under Beth definability). In [56], the authors classify the most common expressive DL dialects based on Beth definability property. Unfortunately, to get a DL dialect that is closed under definability, we must sacrifice some expressive power.

The second problem is tougher to solve, because, to the best of our knowledge, there is no complete procedure for rewriting arbitrary FOL formulas into DL (or reporting an error if the translation is impossible). This means that sometimes, even if it is possible to express the interpolant D_i as $\mathcal{SHI}$ concept, the translation will miss it.

In this work, we implemented a specialized FOL-to-$\mathcal{SHI}$ DL translator. It is based on reverse application of the rules in Table 4.1 and certain heuristics. Heuristics are designed based on the structure of the interpolants generated by ITB tool. For our evaluation, we found that this translator performed rather well; there were no queries that we failed to translate from FOL to $\mathcal{SHI}$. Depending on the ontology and types of queries, additional heuristics may be added (or existing ones may be modified) to improve the recall ratio. Of course, this problem can be avoided altogether, if interpolant enumeration tool for description logics is available.

Now, suppose that the translation is successful, and D is a $\mathcal{SHI}$ concept which is a translation of the FOL interpolant D_i. To determine if D is rewritably equivalent to the query concept C, we still need to determine whether $D \in \mathcal{L}_{Q_2}$. In our approach, we simply enumerate through all concepts $D' \in \mathcal{L}_{Q_2}$, and syntactically check whether $D = D'$. If we find such D', we report D as a rewritably equivalent concept to C, and go ahead with rewriting the input query to: $\pi^{\mathcal{K}}_{Pd}(\sigma^{\emptyset}_{D}(S_{j_1} \sqcap \cdots \sqcap S_{j_k}))$. Alternatively, instead of syntactically checking whether $D \in \mathcal{L}_{Q_2}$, we can perform a semantic check: for each $D' \in \mathcal{L}_{Q_2}$, check whether $\mathcal{T} \models D \equiv D'$. This will result in worse performance, compared to syntactic check, however, with semantic check, we can stop enumerating interpolants as soon as we generate an interpolant that is expressible as a $\mathcal{SHI}$ concept.

If the generated interpolant D_i cannot be translated into $\mathcal{SHI}$ concept (either because it is not expressible as $\mathcal{SHI}$ concept, or because the system cannot find a translation), or $D_i \notin \mathcal{L}_{Q_2}$, we request next interpolant in the stream — D_{i+1}, and the entire process, as described above, is repeated for this new interpolant. In this circumstance, we are not aware of any results that could provide stopping conditions for interpolant enumeration. Therefore, interpolant enumeration can run indefinitely, without ever generating a result. A simple, practical solution to this, is to introduce the upper bound on the number of interpolants that could be generated and processed, or provide a specific amount of time for interpolant enumeration. This upper bound can be set experimentally, depending on the application and the underlying ontology.

Chapter 5

Experimental Evaluation

We evaluated the performance of our approach for rewriting step of query evaluation (i.e. step 2 in Figure 4.1) in comparison to the brute force method. For our approach, we used CARE as query answering engine, ITB for enumerating FOL interpolants, and our custom implemented $\mathcal{SHI}$-FOL and FOL-$\mathcal{SHI}$ translators. Checking membership of generated interpolants D_i in $\mathcal{L}_{Q_2}$ was performed using a syntactic concept comparison functionality that is part of CARE by using hash functions. The brute force method was implemented by enumerating all concepts in $\mathcal{L}_{Q_2}$ in linear fashion, until we either find a concept equivalent to the selection condition concept C, or we exhaust all concepts in $\mathcal{L}_{Q_2}$. Subsumption checks for the brute force method were performed with CARE, however, in principle, any other DL reasoner could be used.

Despite potential problems (outlined in Section 4.2) with using the $\mathcal{SHI}$ DL dialect and FOL, in our experiments, we did not observe any errors due to failure to generate an interpolant that can be expressed as a $\mathcal{SHI}$ concept, nor errors due to failure to translate FOL formula to $\mathcal{SHI}$ concept.

We are not aware of any benchmark ontologies that would be suitable for our purposes. Therefore, we conducted experiments on two data sets: LUBM — a benchmark ontology for query answering ([29]), and LUBMMOD — an ontology that we created manually by adding extra axioms to the TBox of LUBM. For each ontology, we created ten test queries, with a set of relevant cached query results for each query. Sets of relevant cached query results were manually synthesized. Conceptually, one may imagine that a set of cached results $S = \{S_1, \ldots, S_n\}$ exists for CARE, and we run a procedure, like the one in [26], to generate a set of relevant cached results for each query.

For each ontology, we measured and compared query rewriting times using our approach versus the brute force method. Also, we measure the time share of interpolation in query rewriting with our approach. Since ITB is not optimized for DL reasoning, we did not expect our approach to outperform the brute force method. However, we expected the performance be comparable. To evaluate a possible benefit of using an interpolant enumeration tool specific for DLs, we simulated interpolation on a DL reasoner which is part of CARE. In particular, from the original knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we created a new TBox $\mathcal{T}^*$ which is a copy of $\mathcal{T}$ except that, we replace every concept C_i that appears in $\mathcal{T}$ and that does not appear in $\mathcal{L}_{Q_2}$ by C_i^*. In a similar fashion, we created a new query concept C^* from the original query C. Then, we ran the following subsumption check on CARE: $\mathcal{T} \cup \mathcal{T}^* \models C \sqsubseteq C^*$. From the tableaux expansion for this subsumption check, one can generate an interpolant in time that is linear in the size of the tableaux.

For each experiment, the time for rewriting a query was averaged over 20 independent runs, with outliers disqualified. We deemed a data point as "outlier" if its value was at least one order of magnitude greater than values of the majority of data points. Outliers occurred due to the interference from other processes that ran at the same time as our experiments. The experiments were conducted on a single machine with the CPU with 2

cores, 3.06 GHz and 4 GB of memory. The data sets and results for each experiment are summarized in the subsequent sections.

5.1 LUBM Benchmark

5.1.1 Experiment Setup

For the first experiment, we used LUBM benchmark ontology, with $17K$ individuals in the ABox. There is one domain dependent axiom in the TBox, which we disqualified for our experiments. LUBM benchmark includes 14 test queries. Out of these, only 12 can be expressed as algebraic queries for CARE, and only 8 can be used for our purposes, with minimal modifications (described below). The reason why the other 4 are not applicable, is that in order for the interpolant generation to work for them, we would need to have those specific queries cached, and no others; these cases are trivial, and are covered by experiments with other test queries. With all this in mind, we selected the following test queries, which represent selection conditions of algebraic user queries that may be submitted to CARE, for our evaluation:

Q1: $GraduateStudent \sqcap \exists TakesCourse.Course$

Q2: $Publication \sqcap \exists PublicationAuthor.AssistantProfessor$

Q3: $Professor \sqcap \exists WorksFor.University$

Q4: $Person \sqcap \exists MemberOf.University$

Q5: $Student$

Q6: $Student \sqcap \exists Advisor.Faculty \sqcap \exists TakesCourse.Course$

Q7: $Student \sqcap \exists TakesCourse.GraduateCourse$

Q8: $ResearchGroup \sqcap \exists SubOrganizationOf.University$

Q9: $Person \sqcap \exists MastersDegreeFrom.University$

Q10: $UndergraduateStudent$

Queries Q1-Q5, Q7, Q8, Q10 were taken from the LUBM benchmark, while Q6 and Q9 were manually created to resemble the format and complexity of the other LUBM queries.

As was mentioned above, we needed to modify the original LUBM benchmark queries to be usable with CARE. In particular, queries that used individual names in their formulation, were modified to use the appropriate concepts instead. For example, Q2 originally looked like this:

$$Q := Publication \sqcap \exists PublicationAuthor.\{http://\ldots/AssistantProfessor0\}$$

and we modified it to be:

$$Q := Publication \sqcap \exists PublicationAuthor.AssistantProfessor$$

The reason for this, is that CARE does not support the use of nominals, which means that these types of queries are not handled. One possible work-around for this, as described in [46], is to create a fresh concept name A', and add $A'(http://\ldots/AssistantProfessor0)$ to the ABox; then, we could rewrite the query as:

$$Q := \exists PublicationAuthor.A'$$

For our purposes, this is still not acceptable, because the only way to generate an interpolant for a query that uses such fresh concept A', is to have A' among cached query results. Conceptually, this means that the same type of query, about the same individual has been asked in the past. Thus, we chose to replace individual names in queries by the underlying concepts, without reducing the complexity of the query.

For each user query, we manually synthesized a set of relevant cached query results. Each cached query result S_i was obtained from query (C_i, Pd_i). Since the experiments are used as a proof of concept, for simplicity, we set the query for each S_i to be of the form $(C_i, C_i?)$. The following table shows C_i for each S_i:

Cached Query Results:	
For Q1:	$Person$
	$\exists TakesCourse.GraduateCourse$
	$\exists TakesCourse.Course$
	$GraduateStudent$
For Q2:	$\exists PublicationAuthor.AssistantProfessor$
	$\exists PublicationAuthor.Faculty$
	$Publication$
	$\exists PublicationAuthor.Person$
For Q3:	$\exists WorksFor.University$
	$Professor$
	$\exists WorksFor.Organization$
	$\exists MemberOf.Organization$
	$Person$
For Q4:	$\exists MemberOf.Organization$
	$Person$
	$\exists MemberOf.University$

For Q5:	$Person$
	$\exists TakesCourse.Course$
For Q6:	$Student$
	$\exists TakesCourse.Course$
	$\exists Advisor.Faculty$
For Q7:	$Student$
	$\exists TakesCourse.GraduateCourse$
	$\exists TakesCourse.Course$
For Q8:	$ResearchGroup$
	$Organization$
	$\exists SubOrganizationOf.University$
	$\exists SubOrganizationOf.Organization$
For Q9:	$Person$
	$\exists DegreeFrom.Organization$
	$\exists MastersDegreeFrom.University$
	$\exists DegreeFrom.Organization$
For Q10:	$Person$
	$Student$
	$UndergraduateStudent$

Our main concern when synthesizing the sets of cached query results was to ensure that

these sets are sufficient to generate interpolants for the corresponding queries. Once this goal was achieved, we augmented the sets with other relevant cached query results, in an attempt to complicate the task for ITB and for the brute force method.

5.1.2 Results

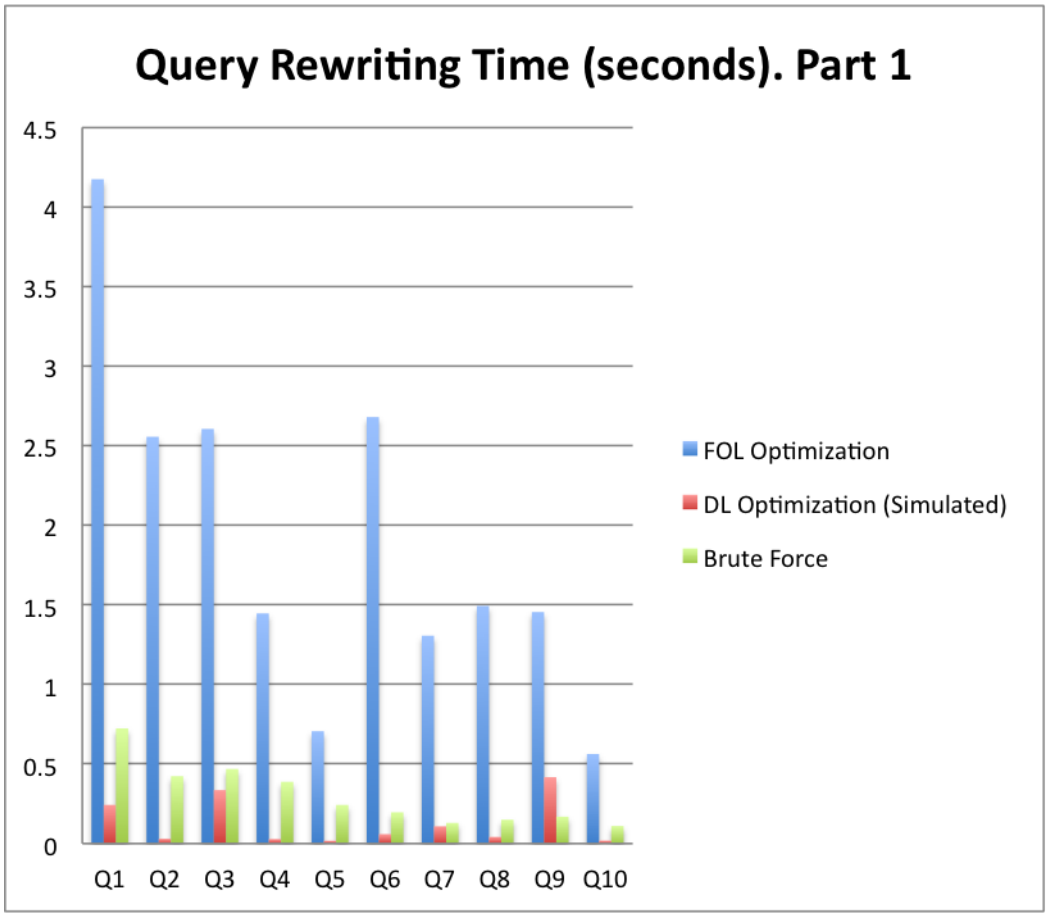

Figure 5.1: FOL and DL based optimization versus the brute force method (LUBM benchmark).

In Figure 5.1, we compare the time (y-axis), in seconds, for rewriting each query in the test suite (x-axis) in order to eliminate $\mathcal{K}$, using our optimization with FOL based interpolant enumeration tool (labeled "FOL optimization") versus the brute force method. We also report expected performance of our optimization with DL based interpolant enumeration (labeled "DL optimization").

Expected time for DL optimization was computed based on the time to simulate interpolation on CARE DL reasoner, with the formula $t_E = 2 \times t_I + t_S$, where t_I represents the time to simulate interpolant extraction on CARE, and t_S represents the time to check whether generated interpolant belongs to $\mathcal{L}_{Q_2}$. We put a factor of 2 in front of the time for simulating interpolation on CARE, because simulation only generates a tableaux proof required to extract interpolants without generating an actual interpolant. Extracting interpolants from the tableaux tree runs in linear time in the size of the tableaux. Therefore a factor of 2 is a sensible choice. Time to determine whether generated interpolant belongs to $\mathcal{L}_{Q_2}$ was taken from the evaluation of the FOL optimization. The reason for this is that we could not get these times for DL optimization directly, since actual interpolants were not extracted in the DL case. However, checking whether an interpolant belongs to $\mathcal{L}_{Q_2}$ is performed with hash functions, implemented for the $\mathcal{SHI}$ dialect on CARE, independently from the interpolant extraction. Therefore the values taken from FOL optimization provide a reasonable approximation.

As expected, FOL optimization did not outperform the brute force method. The reason for this, as mentioned before, is the generality of ITB: it is not optimized for description logics, nor to the situations where many cached results may be relevant to the input query. For this test run, brute force method performs very well for each query. This can be explained by the simplicity of the queries and triviality of the relevant parts of the underlying ontology, which leads to very fast subsumption checks in the brute force

method. Also, the number of relevant cached query results in this experiment is quite small (never more than 5), which bounds the size of $\mathcal{L}_{Q_2}$ by 32. Of course, as the number of relevant cached query results grows, brute force method will become slower (theoretically — exponentially slower). DL optimization, on the other hand, performed rather well, even though CARE reasoner does not implement many optimizations, and, similarly to ITB, is not optimized at all to the situations where many cached query results may be relevant to the user query. DL optimization runs much faster than FOL optimization, and outperforms brute force method for all but one query (Q9). Also, for Q3 and Q7, we observe that expected performance of DL optimization, although slightly better, is close to that of the brute force method. For the rest of the queries, DL optimization runs considerably faster than brute force method (by approximately one order of magnitude). Possible explanation for slower performance of DL optimization for Q3, Q7 and Q9 is a relatively more complicated structure of LUBM ontology in the places that are relevant to these queries (for example presence of inverse roles for Q3 and Q9, or equivalences and more involved concept hierarchy of *Student* and *Course* concepts for Q7). Although we measured only expected performance of DL optimization, generated results showed that using DL reasoner for enumerating interpolants is a sensible next step to make our optimization competitive with the brute force method.

FOL optimization performs substantially worse than brute force for most of the queries; only the times for Q5 and Q10 are comparable to brute force method. This can be explained by relative triviality of Q5 and Q10 compared to other queries in the test suite. Both of them are atomic concepts, and even though there is an equivalence related to Q5 in the LUBM schema, the set of cached results that must be available in order for the interpolants to exist is trivial for both of these queries.

For this experiment, ITB successfully generated interpolants for all queries (similarly,

simulation of interpolation on CARE was successful for all queries), and all interpolants were successfully translated to $\mathcal{SHI}$ concept. For all queries, but Q8, exactly one interpolant was needed in order to generate a rewriting. For Q8, two interpolants were generated. The first interpolant was successfully translated into $\mathcal{SHI}$ concept D', however, $D' \notin \mathcal{L}_{Q_2}$, and so the next interpolant on the stream had to be processed. Concept D' — translation of the first interpolant, contained disjunction and negation, so it is natural that syntactic check for membership of D' in $\mathcal{L}_{Q_2}$ failed. Note, if instead we used semantic method to check whether $D' \in \mathcal{L}_{Q_2}$, it would have succeeded and we would not have to generate the second interpolant.

Another interesting parameter of the experiment to consider, is the order in which $D_i \in \mathcal{L}_{Q_2}$ were enumerated for both, brute force method, as well as for the syntactic membership check of interpolants in $\mathcal{L}_{Q_2}$. Since in brute force method, and in syntactic check for membership of interpolants in $\mathcal{L}_{Q_2}$, we stop enumeration of concepts in $\mathcal{L}_{Q_2}$ as soon as we find $D_i \in \mathcal{L}_{Q_2}$ such that $\mathcal{T} \models D_i \equiv C$, for brute force method, or D_i is the same as the supplied interpolant, for our approach, these procedures are affected by the position of this D_i in the enumeration order of $\mathcal{L}_{Q_2}$. During the evaluation, we, indeed, observed the performance of brute force worsen when the needed concept D_i was at the end of the $\mathcal{L}_{Q_2}$ in the enumeration order. This is easily explained by the extra subsumption checks that need to be performed. At the same time, we found that changing the order of enumeration of $\mathcal{L}_{Q_2}$ had minimal effect on our approach. This is due to the fact that syntactically checking whether an interpolant belongs to $\mathcal{L}_{Q_2}$ is very fast (implemented with hash functions), and, thus, has a negligible impact on the runtime of our optimization. Also, ITB heuristics for selecting a predicate from A_p (which was generated from $\mathcal{L}_{Q_2}$) do not take into account the order in which predicates in A_p are organized. Even though manipulating the order of enumeration of $\mathcal{L}_{Q_2}$ can be used to influence performance of brute force method, in general,

it is impossible to know the best order of enumeration without performing reasoning that is also required for subsumption checks in brute force method. For our experiments, the order of enumeration of $\mathcal{L}_{Q_2}$ was the same for the FOL optimization and for the brute force method; it was randomly set prior to conducting the experiment.

5.1.3 Interpolation Time

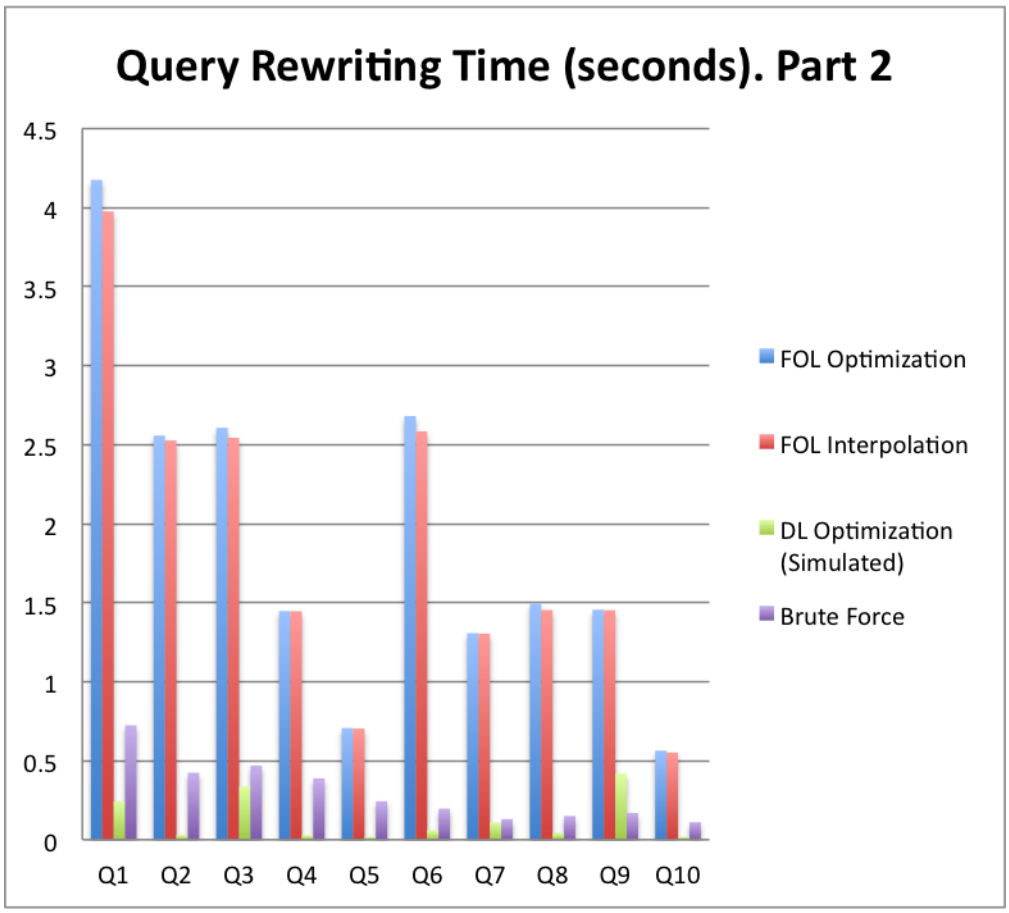

Figure 5.2: Query rewriting time vs. interpolation time (*LUBM* benchmark).

In Figure 5.2, we compare time (y-axis) to enumerate interpolants, in seconds, in the suggested FOL optimization with the DL optimization (described in Section 5.1.2) as well as the brute force method, for each query in the test suite (x-axis). As expected, interpolation time dominates in the FOL optimization.

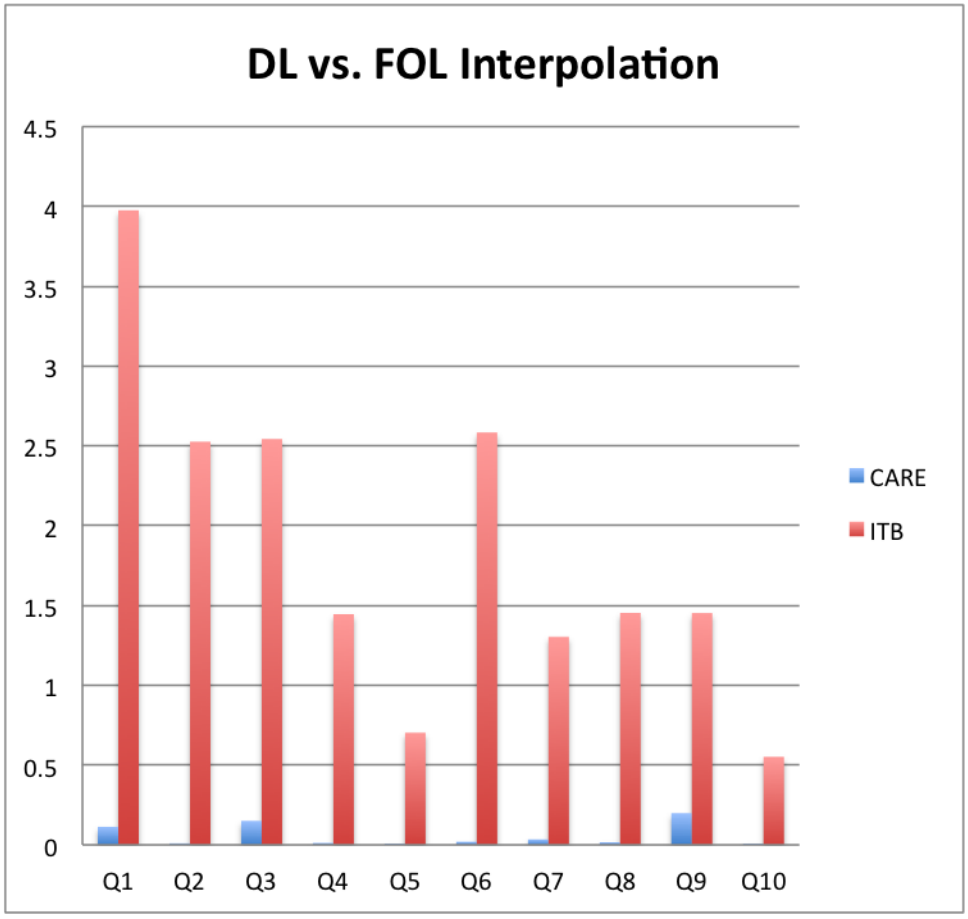

Figure 5.3: Measuting interpolation time on ITB vs. simulation of interpolation on CARE.

Second biggest time share is taken up by FOL to $\mathcal{SHI}$ and $\mathcal{SHI}$ to FOL translation, however it is much smaller than interpolation time, and in most cases is completely negligible. Only for Q1, Q3 and Q6 translation time is noticeable. This can be explained by the

size and complexity of the generated interpolants, since translation time is directly proportional to those two factors. Finally, time to determine whether an interpolant belongs to $\mathcal{L}_{Q_2}$ is negligible in contrast to the first two — any noticeable difference between the FOL optimization time and interpolant enumeration time is due to translation time.

These results reinforced our hypothesis that generality of ITB will cause a slowdown in our approach, and thus it may be worthwhile to explore interpolant extraction/enumeration on specialized DL reasoners.

In Figure 5.3, we report the time (y-axis), in seconds, for extracting interpolants with ITB and simulating interpolant extraction with CARE, for each query of the test suite (x-axis).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
CARE	0.113	0.0096	0.151	0.012	0.0068	0.0197	0.034	0.016	0.199	0.002
ITB	3.976	2.53	2.54	1.445	0.703	2.584	1.304	1.453	1.452	0.552

Table 5.3: DL vs. FOL interpolation time in seconds (original *LUBM*).

Table 5.3 shows the exact times of interpolation with ITB and simulation of interpolation with CARE. The values for simulating interpolation on CARE from Table 5.3 were used to calculate expected performance of DL optimization in section 5.1.2. One can see that interpolation with ITB is slower by one or two orders of magnitude compared to the simulation on CARE. These results show that implementing interpolant extraction on specialized DL reasoner is very promising for our optimization.

From theoretical point of view, since interpolants are generated from a tableaux proof, high worst case complexity of reasoning tasks for many expressive DL dialects suggests

that in some cases interpolant extraction can have a larger impact, than results reported in Figure 5.3. On the other hand, despite high worst case complexity of reasoning in many expressive DLs, there are many known optimizations that work rather well for many practical cases [35]. Of course, this means that interpolant extraction and enumeration is largely dependant on the reasoner on which it is implemented; more precisely, interpolant enumeration is highly dependant on the optimizations and heuristics used in the underlying DL reasoner.

5.2 LUBMMOD ontology

Ontology in the LUBM benchmark is rather simple. Due to this, in the first experiment, we saw rather simple subsumption checks in brute force method and we needed to include individual conjuncts from the benchmark queries in the relevant cached query results in order to be able to generate interpolants. Therefore, for the second experiment, we decided to complicate LUBM ontology, by adding new axioms and concepts to its TBox. The new ontology, LUBMMOD, contains all axioms from LUBM TBox, plus the axioms summarized in the table below; ABox of LUBMMOD is the same as that of LUBM.

Additional Axioms:
$Course \sqsubseteq UndergradCourse \sqcup GraduateCourse$
$GraduateCourse \sqsubseteq Level6 \sqcup Level7 \sqcup Level8$
$Level6 \sqsubseteq GraduateCourse$
$Level6 \sqsubseteq LectureBased$
$Level7 \sqsubseteq GraduateCourse$
$Level8 \sqsubseteq GraduateCourse$
$Level8 \sqsubseteq Seminar$

$Level6 \sqsubseteq (\neg Level7)$

$Level6 \sqsubseteq (\neg Level8)$

$Level7 \sqsubseteq (\neg Level8)$

$Seminar \sqsubseteq Level7 \sqcup Level8$

$Level7 \sqsubseteq LectureBased \sqcup Seminar$

$LectureBased \sqsubseteq Level6 \sqcup Level7 \sqcup UndergradCourse$

$LectureBased \sqsubseteq (\neg Seminar)$

$UndergradCourse \sqsubseteq Course$

$UndergradCourse \sqsubseteq LectureBased$

$UndergradCourse \sqsubseteq (\neg GraduateCourse)$

$Student \sqsubseteq UndergraduateStudent \sqcup GraduateStudent$

$UndergraduateStudent \sqsubseteq (\neg GraduateStudent)$

$Employee \sqsubseteq Faculty \sqcup AdministrativeStaff$

$Faculty \sqsubseteq (\neg AdministrativeStaff)$

$Faculty \sqsubseteq PostDoc \sqcup Faculty \sqcup Professor$

$Professor \sqsubseteq (\neg Lecturer)$

$Professor \sqsubseteq (\neg PostDoc)$

$Lecturer \sqsubseteq (\neg PostDoc)$

$$Professor \sqsubseteq AssociateProfessor \sqcup$$
$$AssistantProfessor \sqcup$$
$$FullProfessor \sqcup$$
$$VisitingProfessor \sqcup$$
$$Chair \sqcup$$
$$Dean \sqcup$$

$$AssociateProfessor \sqsubseteq (\neg FullProfessor)$$
$$AssociateProfessor \sqsubseteq (\neg AssistantProfessor)$$
$$AssistantProfessor \sqsubseteq (\neg FullProfessor)$$
$$Chair \sqsubseteq FullProfessor \sqcup AssociateProfessor$$
$$Dean \sqsubseteq FullProfessor \sqcup AssociateProfessor$$

$$PublishedAuthor \equiv Person \sqcap \exists PublicationAuthor.\top$$

To make the ontology more complicated, we added cover constraints for few concepts in the LUBM ontology. The motivation for doing so came from the observation that disjunction is often poorly handled by reasoners. Each group of axioms in the table above (groups of axioms are separated by blank lines), complicates the structure of the original LUBM ontology in different places. The first group adds a complicated hierarchy of courses that students may take, and various relationships between courses. Second, third and fourth groups of axioms add cover constraints to student, employee and professor concepts, respectively. Finally, the last axiom simply adds another equivalence to the ontology. Our hope was that presence of disjunctions, negations and complex structures in certain parts of the ontology, will burden the subsumption tasks more than interpolant extraction.

5.2.1 Experiment Setup

Since we were not using any known benchmarks for the second experiment, we had to manually synthesize the queries over LUBMMOD ontology. The following ten queries were synthesized, to target the parts of LUBMMOD ontology that were created by the additional axioms. Same as with the first experiment, these represent selection conditions of user queries that are submitted to CARE. Some of these queries, we decided to make

more complicated in terms of structure, compared to the first test suite. In particular, by including disjunction and negations, due to the hypothesis that these constructs may cause complications for reasoners.

Q1: $Professor$

Q2: $Seminar$

Q3: $Faculty$

Q4: $Employee$

Q5: $Course \sqcap Seminar \sqcap Level8$

Q6: $Employee \sqcap (\neg AdministrativeStaff)$

Q7: $GraduateCourse \sqcap (\neg(Level7 \sqcup Level8))$

Q8: $UndergradCourse$

Q9: $Student \sqcap TeachingAssistant \sqcap (\neg GraduateStudent)$

Q10: $Student \sqcap \exists Advisor.Professor$

From the above queries, Q1, Q3, Q4, Q6 target the employee hierarchy of LUBMMOD ontology. Queries, Q2, Q5, Q7, Q8 target the course hierarchy of the ontology. Finally, Q9 and Q10 are of the same style as queries in the first experiment, however they indirectly target student and employee hierarchy in LUBMMOD.

Similar to the first experiment, we had to generate relevant cached query results for each query above. Once again, for simplicity we set the query for each cached query result S_i to be $(C_i, C_i?)$. Selection concepts C_i for each S_i are summarized in the following table.

Cached Query Results:	
For Q1:	$Faculty \sqcap (\neg PostDoc) \sqcap (\neg Lecturer)$
For Q2:	$GraduateCourse \sqcap (\neg LectureBased)$

For Q3:	$Lecturer \sqcup Professor \sqcup PostDoc$
For Q4:	$\exists WorksFor.Organization$
	$Person$
For Q5:	$GraduateCourse$
	$Level8$
For Q6:	$Lecturer \sqcup Professor \sqcup PostDoc$
For Q7:	$Level6$
For Q8:	$LectureBased \sqcap (\neg GraduateCourse)$
For Q9:	$Person$
	$UndergraduateStudent$
	$Student$
	$\exists TeachingAssistantOf.\top$
	$\exists TeachingAssistantOf.Course$
For Q10:	$Student$
	$\exists TakesCourse.Course$
	$\exists Advisor.\top$
	$\exists Advisor.Professor$
	$\exists Advisor.Faculty$

Same as before, we had to make sure that we include sufficient cached query results, to ensure that interpolants can be extracted for the test queries, and then augment the sufficient sets with other relevant cached query results. In addition, however, we attempted to make the relevant cached query results non-trivially related to the corresponding queries, so that more complex parts of LUBMMOD TBox would need to be explored in order to

find an interpolant, or perform a subsumption check. This differs from the set up of the first experiment, where, in most cases, the reasoner only needed to consider concepts that were in $\mathcal{L}_{Q_2}$, or trivial parts of LUBM ontology, in order to generate interpolants or perform subsumption checks.

5.2.2 Results

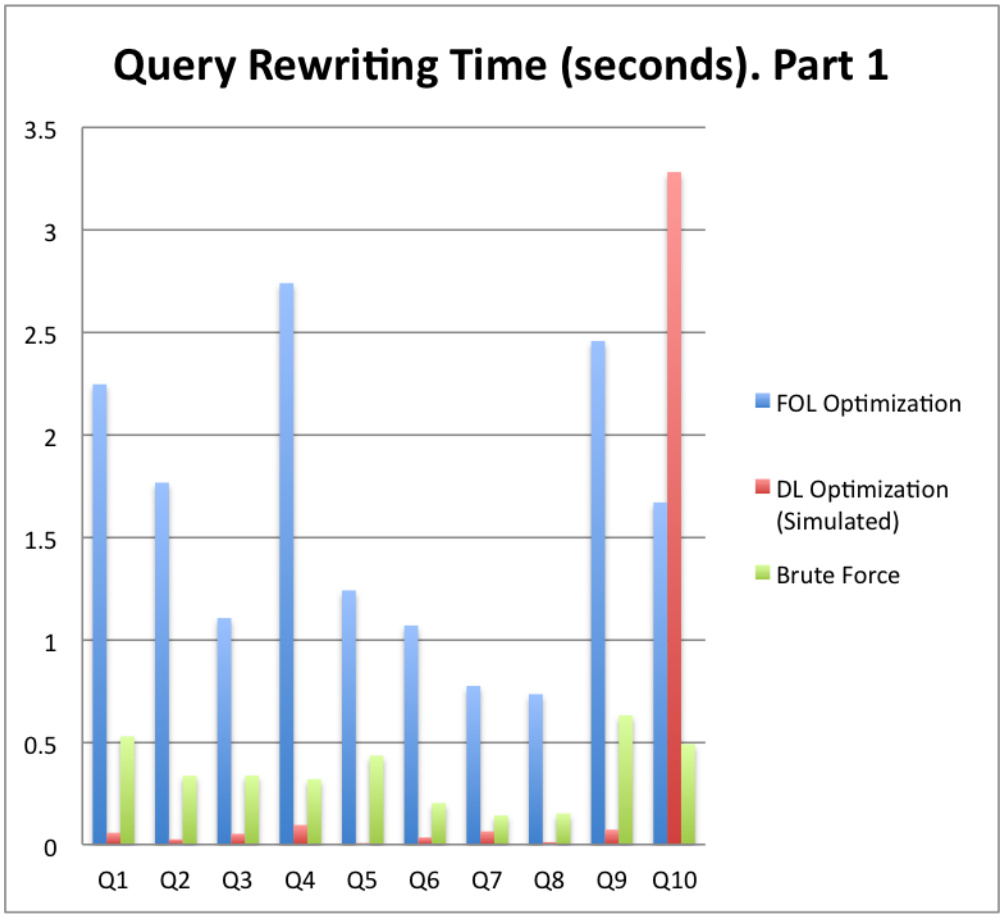

Figure 5.4: FOL and DL based optimizations versus the brute force method (LUBMMOD ontology).

In Figure 5.4 we report the time (y-axis), in seconds, of rewriting all queries in the second test suite (x-axis) in order to eliminate reasoning with respect to $\mathcal{K}$, using proposed optimization with FOL based interpolant enumeration tool (labeled "FOL optimization"), using proposed optimization with DL based interpolant enumeration tool (labeled "DL optimization") and the brute force method. For DL optimization we report expected runtime (calculated by the same formula as in the first experiment).

Same as in the first experiment, FOL optimization does not outperform the brute force method, which is still an expected outcome. The size of $\mathcal{L}_{Q_2}$ remained small for all queries in the second experiment. However the subsumption checks performed by the brute force method were more complicated, compared to the first experiment. This explains why times for brute force method increased by a small amount relatively to the first experiment. One can note that the gap between performance of the FOL optimization and the brute force method shrunk. A possible explanation for this is that additional constraints in the LUBMMOD TBox had a, relatively, smaller effect on the interpolant enumeration with ITB tool compared to subsumption checks with CARE.

DL optimization still performs rather well compared to FOL optimization and the brute force, for all queries except Q10. Compared to results of the first experiment, we see that the average (excluding outlier — Q10) gap between performance of the DL optimization and the brute force method increased. Comparing results of the first (Figure 5.1) and second (Figure 5.4) experiments we can see that physical times for brute force method and DL optimization for queries in the second test suite, decreased or remained the same. This means that the improvement of the performance of DL optimization is relatively greater than that of the brute force method. Since for the second experiment, reasoning tasks required deeper exploration of the structure of the underlying ontology, and the actual ontology LUBMMOD is more complex than LUBM, this may be a sign that our approach

will perform better than the brute force method in cases when extensive exploration of complex TBoxes is required.

The brute force method performed much better than the FOL optimization for Q1, Q2, Q4 and Q9. A possible explanation for this may be that since the underlying ontology became more complicated, the interpolants generated by ITB became more complex, and so translation time and time to check for membership of the generated interpolants in $\mathcal{L}_{Q_2}$ increased. More detailed discussion of this hypothesis follows in section 5.2.3. DL optimization considerably outperforms FOL optimization (by approximately 1-2 orders of magnitude), and the brute force method (by approximately one order of magnitude), for all but one query (Q10). For Q7, DL optimization is close to brute force method, however this can be explained by the simplicity of the query. Indeed, brute force method performs the best on Q7 and for FOL optimization, it is second best performance, out of the entire test suite of queries. For Q10, we observe that DL optimization performs much worse than both, FOL optimization and brute force method. The main reason for this is high complexity of simulating interpolation on CARE for Q10 (> 1.5 seconds); once again, a more detailed discussion of this occurrence follows in the next section.

For the second experiment, we did not observe any repeated generation of interpolants due to failure to translate FOL formula to $\mathcal{SHI}$ concept. For all queries, except Q9, exactly one interpolant was generated to create the rewriting. For Q9, three interpolants were generated, in order to produce a rewriting. The first two interpolants contained disjunctions and negations, and therefore failed the syntactic membership check in $\mathcal{L}_{Q_2}$.

Same as in the first experiment, we observe that order of enumeration of concepts in $\mathcal{L}_{Q_2}$ had a negative effect on the performance of the brute force method. However, this effected was relatively smaller compared to the first experiment, since the size of $\mathcal{L}_{Q_2}$ for the queries in the second test suite is, on average, smaller than the size of $\mathcal{L}_{Q_2}$ for the

queries in the first test suite. Effects of changing the order of enumeration of concepts in $\mathcal{L}_{Q_2}$ on FOL optimization remained negligible. This was expected behaviour, for the same reasons as in the first experiment.

5.2.3 Interpolation time

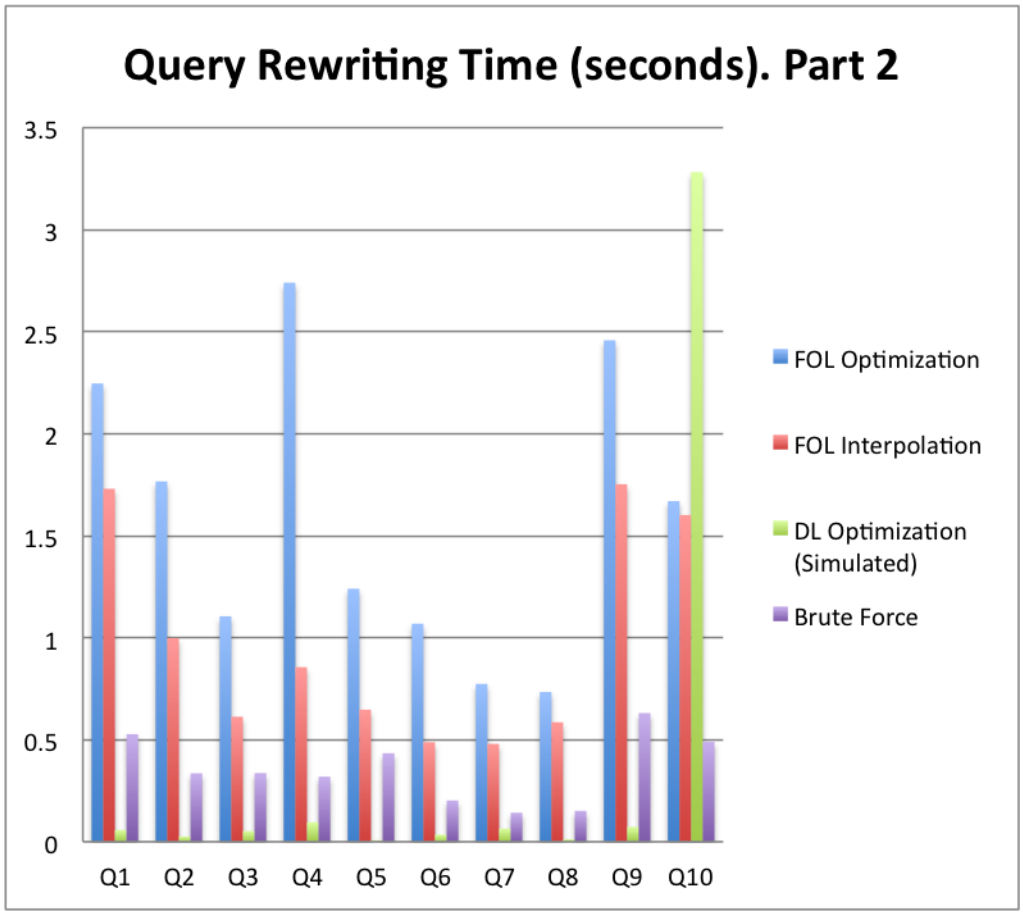

Figure 5.5: Query rewriting time vs. interpolation time (LUBMMOD).

Same as with the first experiment, we timed just the interpolant enumeration portion of the FOL optimization. In Figure 5.5, we show the time (y-axis), in seconds, of interpolant enumeration, compared to complete rewriting with the FOL optimization, DL optimization and the brute force method, for each query in the test suite (x-axis).

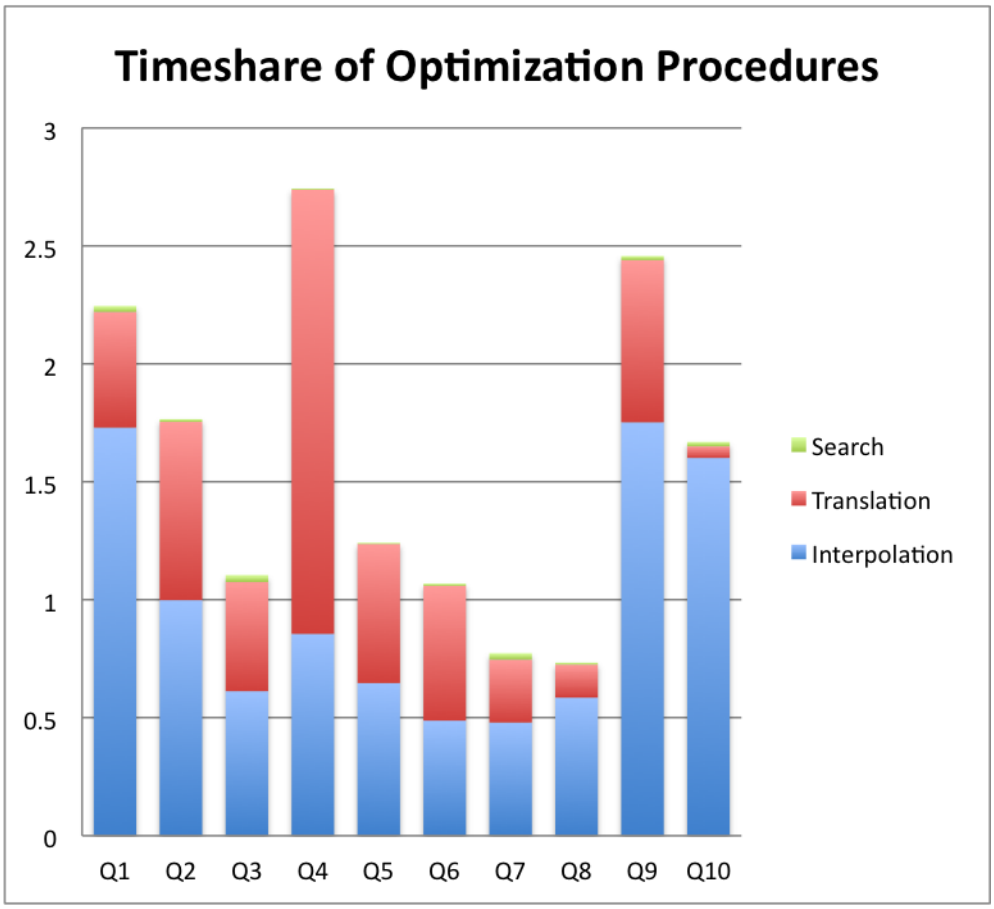

Figure 5.6: Time share of interpolant enumeration, translation and syntactic check of membership of interpolant in $\mathcal{L}_{Q_2}$ (*LUBMMOD* ontology).

For the second data set, we observe an improvement in interpolant enumeration time, relatively to the time taken by the entire FOL optimization procedure. Indeed, in the

first experiment, we saw that FOL optimization was completely dominated by interpolant enumeration; FOL to $\mathcal{SHI}$ and $\mathcal{SHI}$ to FOL translation and checking the membership of the interpolant in $\mathcal{L}_{Q_2}$ were negligible in comparison (Figure 5.2).

In this case, we observe that while checking membership of the interpolants in $\mathcal{L}_{Q_2}$ still occupies a negligible share of the time of the entire FOL optimization, translation part of the procedure takes substantially more time. Figure 5.6 shows time shares (y-axis), in seconds, of the separate parts of the proposed FOL optimization for each query in the test suite (x-axis). On can see that translation time significantly increased compared to the first experiment. For Q2, Q3, Q5, Q6, it takes up approximately half of the time of the entire optimization procedure, and for Q1, Q8 and Q9 — approximately a third of the time is taken by translation. Finally, in Q4 — translation time dominates interpolant enumeration, occupying almost 70% of the time of the FOL optimization procedure.

A decrease in the interpolant enumeration time can be explained by smaller number of relevant cached query results for most queries in the second experiment, compared to the first one. This indirectly decreases the number of unsuccessful tableaux expansions performed by ITB to generate an interpolant, and thus reduces the interpolant generation time. Only with Q10 we see that almost entire optimization time is taken up by interpolant enumeration, while translation time is negligible. The format of Q10 as well as of the relevant cached query results for Q10 are very similar to formats of queries in the first experiment. Essentially, reasoning required to be done by ITB in order to extract an interpolant for Q10 is quite similar to reasoning needed for extracting interpolants in the first experiment; the complications that were added to LUBMMOD were not explored by ITB for Q10. This explains consistency of results for Q10 with those in the first experiment.

Increase in translation time for this experiment, can be accounted for by larger, and more complex interpolants produced by ITB. FOL interpolants produced for the queries

in this experiment were in the format that required more effort to be translated into $\mathcal{SHI}$ concept. Also, for queries in this experiment, interpolants contained many unnecessary repetitions of predicates, which had to be removed during translation.

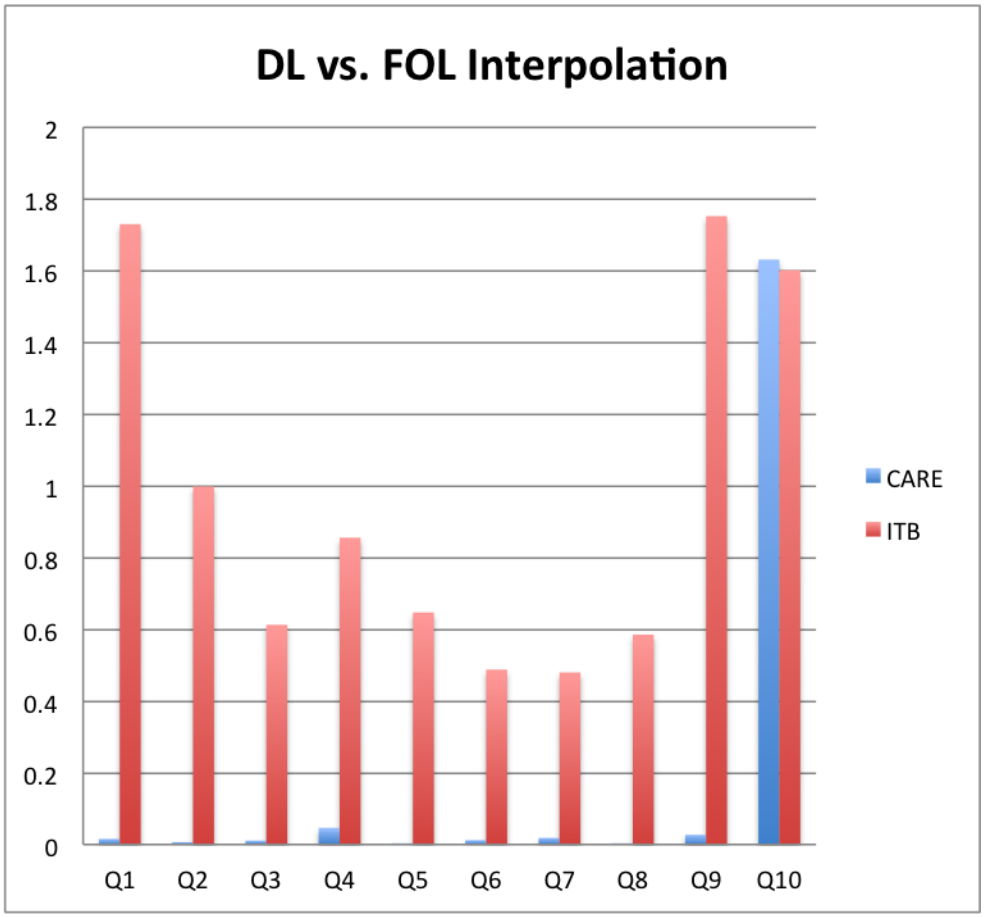

Figure 5.7: Measuting interpolation time on ITB vs. simulation of interpolation on CARE (LUBMMOD).

Increase in the relative time share of translation in our approach suggests another reason why implementing interpolant enumeration on a specialized DL reasoner, may be a sensible next step for our optimization. If interpolant enumeration is a part of a DL reasoner, the

generated interpolants would already be in the needed DL dialect (provided that the DL dialect has definability property), and so translation step would be not necessary.

For this experiment, we also implemented simulation of interpolation on CARE. Figure 5.7 summarizes the time (x-axis), in seconds, for simulating interpolant extraction on CARE, in contrast to interpolant extraction with ITB, for each query in the test suite (y-axis). For Q1-Q9, we continue to observe that simulation on CARE performs better than interpolant extraction on ITB, by at least one order of magnitude. However, for Q10, we see that interpolant generation on ITB is in fact faster than simulation on CARE. This is a rather surprising result; a thorough investigation of the internal structure of CARE reasoner is needed in order to find out the reason for this. One possible explanation is that when performing a subsumption check $\mathcal{T} \cup \mathcal{T}^* \models C \sqsubseteq C^*$ for Q10, CARE needlessly processed parts of LUBMMOD ontology related to courses, employees and professors.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
CARE	0.016	0.007	0.011	0.047	0.002	0.012	0.019	0.002	0.027	1.631
ITB	1.73	0.999	0.613	0.856	0.647	0.488	0.48	0.585	1.753	1.602

Table 5.7: DL vs. FOL interpolation time in seconds (*LUBMMOD* ontology).

Table 5.7 shows the exact values from Figure 5.7. These values were used for calculations of the expected performance of DL optimization in Section 5.2.2.

In general, results of the second experiment give further evidence that implementing interpolant enumeration on a specialized DL reasoner will greatly improve performance of our approach, even to the level of outperforming the brute force method. At the same time, results for Q10, suggest that even when implemented on specialized DL reasoner, in-

terpolant extraction, would require some, potentially application dependent, optimizations in order for it to be competitive in various special cases.

Chapter 6

Conclusion and Future Work

In this chapter we summarize the procedure for our optimization, comment on the results of the initial evaluation of the approach, make some final remarks about the problem and proposed solution, and suggest directions for the future work.

6.1 Summary

In this work, we tackled a problem of improving assertion retrieval over $\mathcal{SHI}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, through eliminating the need to reason with respect to $\mathcal{K}$ for evaluating user queries. In particular, we considered rewriting a selection concept C of the original user query as another concept D, which belongs to $\mathcal{L}$ — a representative language (Definition 6) of cached query results, relevant to C (Definition 16). There is a brute force procedure for doing this; it amounts to enumerating through each concept $D_i \in \mathcal{L}$, and checking whether $\mathcal{T} \models C \equiv D_i$. A concept D_i which satisfies this logical consequence becomes a rewriting for C; if no such D_i was found, we can report that the rewriting is not possible. Although

it does the job, this method can be rather slow, because in the worst case, it performs $|\mathcal{L}|$ number of subsumption check with respect to $\mathcal{T}$ (which is an exponential procedure), and by Definition 6, the size of $\mathcal{L}$ is exponential in the number of relevant cached query results.

In our approach, we use definability and interpolation to find a suitable rewriting for C. We enumerate interpolants of C, with respect to $\mathcal{T}$, from the set of concepts S_p, which is generated from atomic concepts and roles in $\mathcal{L}$, as well as atomic concepts that are introduced as definitions for more complex constructs in $\mathcal{L}$. For each generated interpolant D_i, we check if it belongs to $\mathcal{L}$, and stop the enumeration once we find $D_i \in \mathcal{L}$. We made a conjecture that this approach outperforms the brute force method in many cases.

We conducted experimental evaluation of our approach in contrast to the brute force method. Since we did not have an implementation of the interpolant enumeration tool for DLs, we had to use ITB tool to enumerate interpolants in FOL. Using ITB forced us to implement a translator of $\mathcal{SHI}$ concepts into FOL, and FOL formulas into $\mathcal{SHI}$ concepts. Due to the need for the latter translation, and the fact that we used $\mathcal{SHI}$ DL dialect, which is not closed under definability property, we lost completeness for our approach. However, we did not experience problems with neither interpolation not translation in our experiments. Experimental results showed that our approach with ITB performed worse than the brute force method. The most notable reason for slowness of our approach was due to FOL interpolation and translation of FOL interpolants into $\mathcal{SHI}$ concepts. Due to this, we also ran a simulation of interpolation for a $\mathcal{SHI}$ DL dialect (using CARE reasoner), by executing a tableaux proof required for generating interpolants. We found that this worked well, outperforming ITB by up to two orders of magnitude, and the brute force method by approximately one order of magnitude, for all but one test query. This showed promise in implementing our approach with an interpolant enumeration tool specific to DLs.

6.2 Additional Remarks

Experimental evaluation shows that the total time taken by our optimization with ITB interpolation is at most a few seconds (it would be much less with the interpolant enumeration tool specific to DL). This is a positive result in itself, since there may be situations when it would be impossible to generate a rewriting, perhaps because the set of relevant cached query results is not sufficient. In these circumstances, the time spent on our approach is completely wasted. So, the fact that our optimization does not take up much time relative to full query evaluation, which is in the order of tens and sometimes even hundreds of seconds for the test queries, suggests that it is sensible to use our approach despite the possibility of failing to eliminate $\mathcal{K}$.

Also, it is important to note that our approach should be used in an ad-hoc manner with other optimizations for query evaluation. So, it is quite conceivable that there will be some optimizations that run before ours, which may modify the input to our approach, as well as after our approach. The possible optimizations that come after our step will use our rewriting as input, and so they may provide useful guidelines for the heuristics for interpolant generation (for example they may provide some favourable guidelines on the structure of the interpolants).

Finally, our approach may be adopted to the case where user query is $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C}(Q_1))$, where Q_1 is itself a query of the format $\pi^{\mathcal{K}}_{Pd_1}(\sigma^{\mathcal{K}}_{C_1}(Q_2))$. We can use our approach to produce a rewriting:

$$Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\emptyset}_{D}(Q_1)),\ \text{where}\ \mathcal{T} \models C \equiv D,\ \text{and}\ D \in \mathcal{L}_{Q_1}$$

and $\mathcal{L}_{Q_1}$ is as per Definition 6. The only modification of the procedure for this case would be eliminating the computation of the smaller set of relevant cached query results.

6.3 Future Work

In this work we conducted only initial study and experimental evaluation of the efficacy of the proposed optimization. Therefore, there are a number of promising directions for future developments. In this section, we provide some details on the possibilities for future work, in both, practical and theoretical directions.

6.3.1 Interpolant Enumeration in DL

Results of our experimental evaluation suggest that the main reason for negative performance of our approach compared to the brute force method is using FOL theorem prover for interpolant enumeration, instead of a specialized DL reasoner. Timing the tableaux proof required for extracting an interpolant on CARE DL reasoner reinforced the conjecture that enumerating interpolants with specialized DL reasoner will greatly improve performance of the proposed optimization. In addition, if a specialized reasoner for some DL dialect $\mathcal{L}$ is used, and $\mathcal{L}$ has Beth definability property, the resulting interpolants will already be in $\mathcal{L}$ dialect, and so there will be no need for the translation step. This can also have a considerable impact on the performance of our optimization, since translation can take up a big share of optimization time (Figure 5.6). Therefore, one of the main vectors for future work, is integrating interpolation into a DL reasoner (for example CARE), and using it for our procedure.

Adding interpolant generation functionality to a tableaux based DL reasoner is not very hard. It amounts to adding special purpose labels to each node of the tableaux tree, which will be used only for interpolant extraction, and not for expansion of the actual tableaux. This additional information is added according to well defined rules, outlined in [56] for some expressive DL dialects. A much harder problem is optimizing interpolant generation,

by, for example, creating some useful heuristics for tableaux expansion. Optimizations will be needed, since for some expressive DLs, the worst case size of interpolants is double exponential [56]. Results of simulating interpolation on CARE, reinforced this observation, since in some cases FOL interpolant generation outperforms simulation of interpolant generation on CARE (Q10 in Figure 5.7).

For our approach, we need to not only generate an interpolant, we need to be able to enumerate interpolants based on some, potentially application dependant, cost metric. So, continuing in this direction of future work, one would look at adopting interpolant enumeration method from [57] and any of the optimizations or heuristics applied to this method in the implementation of ITB, to DL reasoners.

6.3.2 Theoretical Results for Interpolant Enumeration

Some theoretical directions for future work are also possible. In particular, exploring in more detail the closure under Beth definability property for various DL dialects, and researching possible stopping conditions for interpolant enumeration. Both of these research directions would contribute to the completeness results for our approach.

In [56], the authors provide classification of some expressive DL dialects with respect to definability property. It turns out that many expressive description logics lack Beth definability property, or have a weaker concept name Beth definability property. CBP is not sufficient for our purposes, since it forces the set of concepts S, from which interpolants are generated, to contain all role names in the signature of TBox. However, by closer inspection of the examples that break definability property for expressive DLs [56], it looks like this happens in rather obscure circumstances. Therefore, it may be worthwhile to explore and describe more precisely the situations that cause definability property to fail

for various DL dialects, and, perhaps, identify the exact conditions that would guarantee definability to be present in DL dialects. Developments in this direction of research, will contribute to the study of completeness of our approach. For example, we may be able to apply our optimization to more TBoxes, with certainty that if an interpolant exists, it will be generated in the DL dialect in which the TBox is expressed.

Another direction for theoretical research is exploring possible stopping conditions for interpolant enumeration. In this work, we set a hard limit on the number of interpolants that are generated before we terminate the procedure and decide that the rewriting is not possible. Such a limit is application dependent and can be set experimentally. Although this may be an acceptable solution for many practical applications, it is not hard to conceive circumstances when the rewriting is possible, but not enough interpolants were generated to find it. Note, this may arise with both FOL and DL interpolant enumeration, since in both cases, we need to check if the generated interpolants belong to $\mathcal{L}_{Q_2}$. For this reason, it would be beneficial to have a theoretical condition (perhaps on the structure of the tableau trees, or actual interpolants that are generated), such that when it is satisfied, we know for sure that none of the interpolants that we can generate from this point on, would produce an acceptable rewriting for the supplied query. Further, in this line of research, one may look at the theoretical conditions on the structure of the partially generated interpolant, to determine whether continuing the extraction can produce an interpolant that belongs to $\mathcal{L}_{Q_2}$, and if not, we can move on to extract the next interpolant in the enumeration order.

6.3.3 Extensions to the Procedure

Our procedure for rewriting a user query should be used as an ad-hoc optimization in query compilation process. One would expect to encounter many circumstances where

our approach would not produce a rewriting, because relevant cached query results are not sufficient to generate an interpolant. So, another direction for possible future research deals with reducing the number of situations in which the proposed optimization is completely useless.

One possible way how this can be achieved, is by rewriting the query that is used as an input to our procedure. For example, suppose that the user query is $Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C_1 \sqcap C_2}(Q_1))$, where Q_1 can be rewritten in terms of cached query results, and the set of relevant cached query results is not sufficient to produce an interpolant D to replace $C_1 \sqcap C_2$ and replace $\mathcal{K}$ with an empty knowledge base. However, it is sufficient to produce an interpolant D' for C_2. In this case, if we rewrite the input query as:

$$Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C_1}(\sigma^{\mathcal{K}}_{C_2}(Q_1)))$$

we can still use our procedure to partially eliminate reasoning with respect to $\mathcal{K}$:

$$Q = \pi^{\mathcal{K}}_{Pd}(\sigma^{\mathcal{K}}_{C_1}(\sigma^{\emptyset}_{D'}(S_{j_1} \sqcap \cdots \sqcap S_{j_m})))$$

Although we eliminated the need for reasoning with respect to $\mathcal{K}$ only for part of selection condition, it still may improve performance in some cases.

Another, rather obvious, extension of out approach is to attempt to eliminate the need to reason with respect to $\mathcal{K}$ for evaluating projection operator: given a user query $Q = \sigma^{\mathcal{K}}_{C}(\pi^{\mathcal{K}}_{Pd}(Q_1))$, rewrite it as $Q = \sigma^{\mathcal{K}}_{C}(\pi^{\emptyset}_{Pd_1}(Q_1))$.

Our procedure can also be extended to produce better rewritings (i.e. rewritings that will result in better input for subsequent optimization steps, or more efficient query plan). In this work, we stop the enumeration of the interpolants as soon as we find a suitable one (likewise for the brute force method, we stop enumeration of concepts in $\mathcal{L}_{Q_2}$ as soon as we find a concept logically equivalent to the selection condition of the query). Instead, we

could continue to enumerate the interpolants (or enumerate through the concepts in $\mathcal{L}_{Q_2}$ for the brute force method) until we find an interpolant (or concept) which would produce a better query plan. Thus, a possible direction for future research may be to create a cost model for interpolant enumeration. Such cost model should help estimate the cost of the final query plan that would be generated if a given interpolant is used for the rewriting.

References

[1] OWL 2 Web Ontology Language Profiles (Second Edition), 2012. Available at http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[2] Open data initiative, 2013. Available at http://www.opendatainitiative.org/.

[3] Sparql 1.1 entailment regimes, 2013. Available at http://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/.

[4] Sparql 1.1 overview, 2013. Available at http://www.w3.org/TR/sparql11-overview/.

[5] Serge Abiteboul, Richard Hull, and Victor Vianu., editors. *Foundations of Databases*. Addison-Wesley, 1995.

[6] Michael O. Akinde and Michael H. Bohlen. Efficient computation of subqueries in complex OLAP. In *Proceedings of the ICDE Conference*, pages 163–174, 2003.

[7] Nash Alan, Segoufin Luc, and Vianu Victor. Views and queries: Determinacy and rewriting. *ACM Trans. Database Systems*, 35:21:1–21:41, 2010.

[8] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the*

6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference, pages 722–735, 2007.

[9] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* 2003.

[10] Franz Baader and Ulrike Sattler. Tableau algorithms for description logics. *STUDIA LOGICA*, 69, 2000.

[11] Willem E. Beth. Indagationes mathematicae. In *On Padoas method in the theory of definition*, pages 330–339, 1953.

[12] Elena Botoeva, Ro Artale, and Diego Calvanese. Query rewriting in DL-Lite$_{horn}^{(\mathcal{HN})}$. In *Proceedings of 23rd International Workshop on DescriptionLogics*, 2010.

[13] James Brotherston and Rajeev Goré. Craig interpolation in displayable logics. In *Proceedings of the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 88–103, 2011.

[14] Diego Calvanese, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, pages 602–607, 2005.

[15] Edgar F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[16] William Craig. Three uses of the Herbrand-Genzen theorem in relating model theory and proof theory. In *Journal of Symbolic Logic*, page 269285, 1957.

[17] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Li Ma, Edith Schonberg, Kavitha Srinivas, and Xingzhi Sun. Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In *Proceedings of the International Semantic Web Conference*, pages 403–418, 2008.

[18] Melvin Fitting, editor. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag, 1996.

[19] Enrico Franconi, Volha Kerhet, and Nhung Ngo. Exact query reformulation over $\mathcal{SHOQ}$ DBoxes. In *Proceedings of the 2012 International Workshop on Description Logics*, 2012.

[20] Enrico Franconi, Volha Kerhet, and Nhung Ngo. Exact query reformulation with first-order ontologies and databases. In *Logics in Artificial Intelligence - 13th European Conference*, pages 202–214, 2012.

[21] Enrico Franconi, Volha Kerhet, and Nhung Ngo. Exact query reformulation over databases with first-order and description logics ontologies. *Journal of Artificial Intelligence Research*, 48, 2013.

[22] César A. Galindo-Legaria and Milind M. Joshi. Orthogonal optimization of subqueries and aggregation. In *Proceedings of the ACM SIGMOD Conference*, pages 571–581, 2001.

[23] Amlie Gheerbrant and Balder ten Cate. Craig interpolation for linear temporal languages. In *CSL*, volume 5771 of *Lecture Notes in Computer Science*, pages 287–301, 2009.

[24] Giuseppe De Giacomo and Maurizio Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proc. of KR-96*, pages 316–327. Morgan Kaufmann, 1996.

[25] Birte Glimm, Yevgeny Kazakov, Ilianna Kollia, and Giorgos Stamou. Using the TBox to optimise SPARQL queries. In *Proceedings of the 26th International Workshop on Description Logic*, pages 181–196, 2013.

[26] Jonathan Goldstein and Per-Ake Larson. Optimizing queries using materialized views: A practical, scalable solution. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2001.

[27] Rajeev Gor and Linh Anh Nguyen. Exptime tableaux with global caching for description logics with transitive roles, inverse roles and role hierarchies. In *Proceedings of Tableaux 2007*, pages 133–148. Springer, 2007.

[28] RDF Working Group. Resource description framework, 2004. Available at: http://www.w3.org/RDF/.

[29] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3), 2005. Available at `http://swat.cse.lehigh.edu/projects/lubm/`.

[30] Volker Haarslev and Ralf Möller. Optimization strategies for instance retrieval. In *Proc. of the International Workshop on Description Logics*, 2002.

[31] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB journal*, 10:270–294, 2001.

[32] James Hendler. Agents and the semantic web. *IEEE INTELLIGENT SYSTEMS*, 16(2):30–37, 2001.

[33] Stijn Heymans, Li Ma, Darko Anicic, Zhilei Ma, Nathalie Steinmetz, Yue Pan, Jing Mei, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg,

Kavitha Srinivas, Cristina Feier, Graham Hench, Branimir Wetzstein, and Uwe Keller. Ontology reasoning with large data repositories. In *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications*, pages 89–128. Springer, 2008.

[34] Eva Hoogland. *Definability and Interpolation: Model-theoretic investigations.* PhD thesis, University of Amsterdam, Amsterdam, Holland, 2001.

[35] Ian Horrocks. Practical reasoning for very expressive description logics. In *Journal of the Interest Group in Pure and Applied Logics 8*, pages 293–323, 2000.

[36] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 161–180, 1999.

[37] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *PROC. IJCAI 2005*, pages 466–471. Professional Book Center, 2005.

[38] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for client-server database architectures. *The VLDB Journal*, 5(1):035–047, 1996.

[39] Enrico Marchioni and George Metcalfe. Craig interpolation for semilinear substructural logics. *Math. Log. Q.*, 58(6):468–481, 2012.

[40] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. Technical report, W3C, 2004.

[41] Kenneth L. McMillan. Applications of Craig interpolants in model checking. In

TACAS2005: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 3440, pages 1–12. Springer, 2005.

[42] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data complexity of query answering in expressive description logics via tableaux. *J. OF AUTOMATED REASONING*, 41:61–98, 2008.

[43] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 2009. Available at `http://www.w3.org/TR/owl2-overview/`.

[44] Glen N. Paulley, Per-Åke Larson, and Per-Ake Larson. Exploiting uniqueness in query optimization. In *Proceedings of the International Conference on Data Engineering*, pages 68–79, 1994.

[45] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Journal on data semantics x. pages 133–173. Springer-Verlag, 2008.

[46] Jeffrey Pound, David Toman, Grant Weddell, and Jiewen Wu. An assertion retrieval algebra for object queries over knowledge bases. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1051–1056, 2011.

[47] Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. Latest version available as `http://www.w3.org/TR/rdf-sparql-query/`, 2008.

[48] Mariano Rodrguez-muro and Diego Calvanese. Dependencies: Making ontology based data access work in practice. In *Proceedings of AMW*, 2011.

[49] Riccardo Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *Proc. of ESWC 2012*, 2012.

[50] Grigore Rosu and Joseph Goguen. On equational Craig interpolation. 6(1):194–200, 2000.

[51] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, It. A. Lorie, and T. G. Price. Access path selection in a relational database management system. pages 23–34, 1979.

[52] Timos K. Sellis. Intelligent caching and indexing techniques for relational database systems. *Information Systems*, 13(2):175 – 185, 1988.

[53] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Effective query rewriting with ontologies over DBoxes. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 923–929, 2005.

[54] Inanç Seylan, Enrico Franconi, and Jos de Bruijn. Optimal rewritings in definitorially complete description logics. In *Proceedings of the 23rd International Workshop on Description Logics*, 2010.

[55] Balder ten Cate, Willem Conradie, Maarten Marx, and Yde Venema. Definitorially complete description logics. In *Proceedings of the International Conference of Principles of Knowledge Representation and Reasoning*, pages 79–89, 2006.

[56] Balder ten Cate, Enrico Franconi, and Inanç Seylan. Beth definability in expressive description logics. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1099–1106, 2011.

[57] David Toman and Grant Weddell, editors. *Fundamentals of Physical Design and Query Compilation*. Morgan & Claypool, 2011.

[58] Michael Uschold. Where are the semantics in the semantic web. *AI Magazine*, 24, 2001.

[59] Moshe Y. Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing (STOC 82*, pages 137–146, 1982.

[60] Kristofer Vorwerk and Glen N. Paulley. On implicate discovery and query optimization, 2002.

[61] Timo Weithöner, Thorsten Liebig, Marko Luther, and Sebastian Böhm. Whats wrong with OWL benchmarks. In *Proc. of the Second Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006*, pages 101–114, 2006.

[62] Jiewen Wu. *Answering Object Queries over Knowledge Bases with Expressive Underlying Description Logics*. PhD thesis, University of Waterloo, Waterloo, Canada, 2013.

APPENDICES

Appendix A

First Order Predicate Logic

First order logic (FOL) is an expressive formalism for representing facts of the application domain, and reason about those facts. FOL allows to express various statements through *well formed formulas (wff)* — strings of characters, where each character is either a *non-logical parameter*, a *logical parameter*, a *variable* or a punctuation symbol "(" or ")" or "," or ".". These facts are the explicit knowledge about the application domain. Through inference in FOL, one can also find implicit knowledge — information that is implied by the explicit facts. In this thesis, when we refer to FOL, we mean the *first order predicate logic*.

The *non-logical parameters* in FOL consist of disjoint infinite sets of predicate symbols $P = \{P_1, P_2, \dots\}$ and constants $C = \{c_1, c_2, \dots\}$. A set of non-logical parameters present in a given first order language is called a *signature* of the language. Each predicate symbol has an *arity*, a non-negative integer. Predicates of arity 0 are *propositions*.

The *logical parameters* in FOL are the following symbols: $\{=, \exists, \neg, \wedge, \forall, \vee, \rightarrow\}$. *Variables* in FOL are a countably infinite set $V = \{x_1, x_2, \dots\}$ disjoint from the non-logical

parameters.

A well formed formula conforms to the following grammar:

$$
\begin{aligned}
T &::= x \mid c \\
A &::= T_1 = T_2 \mid P(T_1, \ldots, T_n) \\
\phi &::= A \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \exists x.\phi_1 \mid \forall x.\phi_1
\end{aligned}
$$

where x is a variable, c is a constant, ϕ_1 and ϕ_2 are well formed formulas, P is a predicate symbol of arity n.

Free variables are defined for a term t and for a well formed formula ϕ. Free variables of t, $FV(t)$ are defined as $\{x\}$ if t is a variable x, and $FV(t) = \emptyset$ if t is a constant. For a well formed formula ϕ, the free variables, $FV(\phi)$, are:

$\cup_{1 \leq i \leq n} FV(t_i)$	if ϕ is $P(t_1, \ldots, t_n)$ where P is a predicate.
$FV(t_1) \cup FV(t_2)$	if ϕ is $t_1 = t_2$.
$FV(\phi_1)$	if ϕ is $\neg\phi_1$.
$FV(\phi_1) \cup FV(\phi_2)$	if ϕ is $\phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2$.
$FV(\phi_1) - \{x\}$	if ϕ is $\exists x.\phi_1 \mid \forall x.\phi_1$.

A well formed formula ϕ is called a *sentence* if $FV(\phi) = \emptyset$. For a well formed formula ϕ, some variable x and a term t, such that $FV(t)$ does not contain any quantified variables of ϕ, a *substitution of t for x in* ϕ, denoted by $\phi[t/x]$, means syntactically replacing every occurrence of x, in ϕ, by t. Substitutions can be composed. A FOL *theory* Σ is a set of first order sentences over some signature.

Let S be a first order signature (if the signature is clear from the context, we will not mention it). An interpretation $\mathcal{I}$ over a signature S is a pair: $(\triangle^{\mathcal{I}}, (\cdot)^{\mathcal{I}})$, where $\triangle^{\mathcal{I}}$

is a non-empty *domain* of objects and $(\cdot)^{\mathcal{I}}$ is an *interpretation function* mapping every predicate symbol P with arity m to a subset of $(\triangle^{\mathcal{I}})^m$ and every constant symbol c to itself, which means that $c \in \triangle^{\mathcal{I}}$.

For a given interpretation $\mathcal{I}$, we define a valuation $\mathcal{V}$ to be a total function from a set of variables V to $\triangle^{\mathcal{I}}$. In particular, given a variable x, and an object o from the domain, the valuation $\mathcal{V}[x \mapsto o]$ is defined by:

$$\mathcal{V}[x_1 \mapsto o](x_2) = \begin{cases} o & \text{if } \text{“}x_1'' = \text{“}x_2'' \\ \mathcal{V}(x_2) & \text{otherwise} \end{cases} \tag{A.1}$$

An interpretation $\mathcal{I}$ and a valuation $\mathcal{V}$ over $\mathcal{I}$ is a *model* for a well formed formula ϕ, written $\mathcal{I}, \mathcal{V} \models \phi$, if:

$$\begin{aligned} & \phi = \text{"}P(t_1, \ldots, t_n)\text{"} \text{ and } \langle \mathcal{V}(t_1), \ldots, \mathcal{V}(t_n) \rangle \in (P)^{\mathcal{I}} \\ \text{or } & \phi = \text{"}t_1 = t_2\text{"} \text{ and } \mathcal{V}(t_1) = \mathcal{V}(t_2) \\ \text{or } & \phi = \text{"}\neg\phi_1\text{"} \text{ and } \mathcal{I}, \mathcal{V} \not\models \phi \\ \text{or } & \phi = \text{"}\phi_1 \wedge \phi_2\text{"} \text{ and } \mathcal{I}, \mathcal{V} \models \phi_1 \text{ and } \mathcal{I}, \mathcal{V} \models \phi_2 \\ \text{or } & \phi = \text{"}\phi_1 \vee \phi_2\text{"} \text{ and } \mathcal{I}, \mathcal{V} \models \phi_1 \text{ or } \mathcal{I}, \mathcal{V} \models \phi_2 \\ \text{or } & \phi = \text{"}\phi_1 \rightarrow \phi_2\text{"} \text{ and } \mathcal{I}, \mathcal{V} \models \neg\phi_1 \text{ or } \mathcal{I}, \mathcal{V} \models \phi_2 \\ \text{or } & \phi = \text{"}\exists x.\phi_1\text{"} \text{ and } \mathcal{I}, \mathcal{V}[x \mapsto o] \models \phi_1 \text{ for some } o \in \triangle^{\mathcal{I}} \\ \text{or } & \phi = \text{"}\forall x.\phi_1\text{"} \text{ and } \mathcal{I}, \mathcal{V} \models \neg\exists x.\neg\phi_1 \end{aligned}$$

For an interpretation $\mathcal{I}$ and a wff ϕ, we write $\mathcal{I} \models \phi$ if there exists a valuation $\mathcal{V}$ over $\mathcal{I}$, such that $\mathcal{I}, \mathcal{V} \models \phi$. Let Σ be a first order theory. We write $\mathcal{I}, \mathcal{V} \models \Sigma$ if $\mathcal{I}, \mathcal{V} \models \phi$, for every $\phi \in \Sigma$. A theory Σ is *satisfiable* if it has a model. A well formed formula ϕ is a *logical consequence* of a theory Σ, written $\Sigma \models \phi$, if and only if, for every interpretation

$\mathcal{I}$ and valuation $\mathcal{V}$ such that $\mathcal{I}, \mathcal{V} \models \Sigma$, we have $\mathcal{I}, \mathcal{V} \models \phi$. FOL is semi-decidable, which means that for arbitrary theory Σ and formula ϕ, determining if $\Sigma \models \phi$ is semi-decidable.

A first order formula ϕ with $FV(\phi) = \langle x_1, \ldots, x_n \rangle$ is *domain independent* if for any two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $P^{\mathcal{I}_1} = P^{\mathcal{I}_2}$ for every $P \in sig(\phi)$, and $\triangle^{\mathcal{I}_1} \subseteq \triangle^{\mathcal{I}_2}$, we have that for every substitution θ over $FV(\phi)$ if $\mathcal{I}_2 \models \phi\theta$, then $\mathcal{I}_1 \models \phi\theta$.

A first order theory Σ and a formula ϕ with $FV(\phi) = \langle x_1, \ldots, x_n \rangle$ can be viewed as a database and a query, respectively. Every atomic predicate $P \in sig(\Sigma)$ corresponds to a table. In relational setting a set of all groundings in Σ of each predicate $P \in sig(\Sigma)$ completely describes P. An answer to a query ϕ is a substitution θ over $FV(\phi)$ that replaces every $x_i \in FV(\phi)$ with a corresponding constant symbol c_i that appears in Σ, such that $\Sigma \models \phi\theta$. Finding all such substitutions θ results in computing *certain answers* for a query ϕ with respect to a theory Σ.